Available online at www.sciencedirect.com

SCIENCE DIRECT

Science of Computer Programming 60 (2006) 221–243

Science of Computer Programming

www.elsevier.com/locate/scico

Composing invariants☆

Michel Charpentier

Department of Computer Science, University of New Hampshire, Durham, NH, United States

Received 12 May 2004; received in revised form 18 April 2005; accepted 26 August 2005
Available online 20 October 2005

Abstract

We explore the question of the composition of invariance specifications in a context of concurrent and reactive systems. Depending on how compositionality is stated and how invariants are defined, invariance specifications may or may not be compositional. This article first examines two classic forms of invariants and their compositional properties. After pointing out what we see as deficiencies of these two kinds of invariants, two new forms are defined and shown to have useful compositional properties that the more classic forms do not enjoy. The last form, in particular, is shown to be well suited to situations where none of the other three is adapted.

Keywords: Reactive and concurrent systems; Formal specification; Temporal logic; Compositional verification; Invariants; Software components

1. Introduction

1.1. Motivation

Invariance properties have long played a fundamental part in the specification and verification of computer systems, especially concurrent and reactive systems. Now that compositional reasoning and reusable, *off-the-shelf* components are receiving much attention in the software engineering community, the question arises of the compositionality of invariants.

The answer to this question is not as straightforward as one may think, because there are different definitions of what it means for a specification to be compositional and, maybe more surprisingly, there are different definitions of what it means for a specification to be an invariant.

In this paper, we focus our attention on two possible definitions of being compositional. One corresponds to a widespread intuition of what it means to compose invariants, namely that if all components of a system satisfy the same invariant, then this system also satisfies this invariant. The other form of composition might look more surprising at first but is indeed used in several formal notations: an invariant is satisfied by a system as long as it is satisfied by at least one component of that system.

☆ A preliminary version of this article appeared in the *12th International Symposium of Formal Methods Europe (FME'2003)*.

E-mail address: charpov@cs.unh.edu.
URL: www.cs.unh.edu/~charpov.

0167-6423/$ - see front matter
doi:10.1016/j.scico.2005.08.001

Depending on how invariants are defined, they satisfy zero, one or both of these composition rules. In this article, we consider four possible definitions of invariants, which we study and compare within the same formal framework. We first focus on two classic forms of invariants ($\mathsf{inv}_{\mathcal{S}}$ and $\mathsf{inv}_{\mathcal{W}}$) as they are traditionally defined on closed systems. For a variety of reasons, which are discussed in Section 4.1, these classic definitions of invariants are not well suited to compositional reasoning. We then introduce a third type of invariant ($\mathsf{inv}_{\mathcal{E}}$), defined with composition in mind. Although it is redefined here in an original way, this third form of invariant has been used in several formalisms to specify and verify components and composed systems. It is well suited to *some* situations in compositional reasoning, but we show, with the help of an illustrative example, that there are many important cases where such an invariant will not work and for which the lack of a suitable type of invariant is a severe detriment to compositional reasoning. We then proceed to the definition of a fourth type of invariant, original to this paper, which we believe to address the drawbacks, from a composition standpoint, of the previous three types.

The context of this work is one of reactive systems modeled as transition systems, as in UNITY [1–3] or TLA [4,5]. The question of invariants in an object-oriented context is briefly discussed in the conclusion. In this article, we do not consider specifications other than invariance properties. In particular, we do not discuss the case of liveness and progress specifications [6]. It has been argued that liveness specifications are inherently more difficult to compose than safety specifications. This question is also discussed in the conclusion.

The end of this introductory section is dedicated to presenting a basic set of notation that is used throughout this article. The remainder of the paper is organized as follows. Section 2 introduces two forms of invariants that are commonly defined on transition systems, independently from composition issues. Section 3 discusses the compositional properties of these two forms of invariants. In Section 4, we explore the question of compositional specifications in general and present elements of a framework dedicated to the study of composition related issues. We put this framework to use in Section 5 where two forms of compositional invariants are defined. A simple example, initiated in Section 2, is carried throughout the paper to illustrate the four kinds of invariants and their compositional properties. Proofs related to the new form of invariants are given in Section 6. Proofs related to classic invariants can be found in the literature [7,2].

1.2. Notation and terminology

Generic systems and components are denoted by capitals letters like F and G. In the context of this article, there are no fundamental differences between systems and components, and the two words are used interchangeably. At an informal level, however, the word *component* conveys the idea that the system is used as a part of a larger system, and the word *system* conveys the idea that a component is itself composite. Parallel composition is denoted by $\|$.

Specifications are predicates on systems; in other words, functions from systems to booleans. Function application is denoted by a dot, which is assumed to be left associative and to have higher precedence than boolean connectives. For instance, $\mathsf{inv}.p.F$ is $(\mathsf{inv}.p).F$ and means that specification $\mathsf{inv}.p$ is satisfied by system F.

Following [8], we rely on an "*everywhere*" operator denoted by square brackets. Given a predicate P, the notation $[P]$ means that P is everywhere true (i.e., $P.x$ is true for all x, or $P.x.y$ is true for all x and y, or ..., depending on the type of P). $\neg[\neg P]$ means that P is not everywhere false or, equivalently, that P is satisfiable. We also use the usual quantifiers $\forall$ and $\exists$ with or without an explicit range, delimited by colons. So, the same predicate can be written as $\forall x : P.x : Q.x$ or $\forall x : (P.x \Rightarrow Q.x)$. Quantifiers are assumed to have lower precedence than boolean connectives, so this last expression can also be written as $\forall x : P.x \Rightarrow Q.x$.

2. Inductive and trace-based invariants

2.1. Transition systems

We assume the systems we reason about are modeled as *transition systems*. Typically, such transition systems are represented as tuples that include, at least, a definition of variables, states, transitions and fairness [6]. Fairness constraints (strong, weak, unconditional, ...) are required to reason about *liveness* (and progress) properties. In this article, we are focusing our attention on invariance specifications, which belong to the class of *safety* properties. As a consequence, fairness issues do not play any role and can be ignored altogether in our discussion. Moreover, when fairness and liveness are not involved, any nonempty set of transitions can be assimilated to a unique

(nondeterministic) transition that encompasses all the transitions from the set. Therefore, in this paper, we rely on (unfair) transition systems defined as 5-tuples. A system F is the tuple $(\mathcal{V}.F, \mathcal{L}.F, \Sigma.F, \mathcal{I}.F, \mathcal{N}.F)$ where:

- $\mathcal{V}$ is a finite set of variables, referred to as *state* variables. They are chosen from a universal vocabulary $\mathcal{D}$.
- $\mathcal{L}$ is a subset of $\mathcal{V}$ of *local* variables, which can be read but not written by other systems.
- Σ is a set of *states*. Each state assigns a value to each variable of $\mathcal{V}$.
- $\mathcal{I}$ is an initial condition that defines the possible initial states of the system. $\mathcal{I}$ is a state predicate (free variables in $\mathcal{V}$), which we assume to be satisfiable.
- $\mathcal{N}$ is a predicate transformer (function from state predicates to state predicates) which represents the *next-state* transition of the system using a *weakest precondition* semantics: a state satisfies $\mathcal{N}.F.q$ if and only if every possible next state satisfies q. A WP function, similar to our $\mathcal{N}$, is used in [7].

  We assume that transition systems always allow for stuttering: it is always possible for the next state of a computation to be the same as the current state. Formally, this means that, for any system F and any state predicate q, $[\mathcal{N}.F.q \Rightarrow q]$: if the next state is guaranteed to satisfy q and stuttering is possible, the current state has to satisfy q as well. We also assume that the predicate transformer $\mathcal{N}.F$ is universally conjunctive for any system F, like any weakest (liberal) precondition transformer [8]. It is not necessarily disjunctive, as $\mathcal{N}.F$ in general represents nondeterministic transitions.

2.2. Inductive invariants

A first kind of invariance specification can be defined directly in terms of a transition system. We call these invariants *strong* or *transition-based* or *inductive*, and denote them with $\mathsf{inv}_{\mathcal{S}}$. We define two types of specification, next and inv:

$$\begin{aligned} \mathsf{next}_{\mathcal{S}}.(p,q).F &\triangleq [p \Rightarrow \mathcal{N}.F.q], \\ \mathsf{inv}_{\mathcal{S}}.p.F &\triangleq [\mathcal{I}.F \Rightarrow p] \wedge \mathsf{next}_{\mathcal{S}}.(p,p).F. \end{aligned} \tag{$1_{\mathcal{S}}$}$$

Informally, $\mathsf{next}_{\mathcal{S}}.(p,q)$ means that whenever an action is fired from a state that satisfies p, the resulting state satisfies q. Similarly, $\mathsf{inv}_{\mathcal{S}}.p$ specifies that p is true in any initial state and is preserved by every atomic transition. Therefore, by induction, p is true in every state. It should be noted that, since $[\mathcal{N}.F.q \Rightarrow q]$ because of possible stuttering, $\mathsf{next}_{\mathcal{S}}.(p,q).F \Rightarrow [p \Rightarrow q]$.

Although this paper focuses on inv specifications, we use next specifications as well, especially in proofs. As we will see in Section 3, the part $[\mathcal{I}.F \Rightarrow p]$ in the definition of inv does not involve any difficulty in terms of composition. The part $\mathsf{next}.(p,p)$ is where the core of compositional issues turns up.

2.3. Trace-based invariants

Instead of relying on the definition of a transition system directly, a second kind of invariance specification, which we call *weak* or *trace-based*, can be defined in terms of the possible computations of such a system. A transition system F can be associated with a subset $\mathcal{O}.F$ of $(\Sigma.F)^{\omega}$ of infinite sequences of states defined as follows. An infinite computation $\sigma = \langle \sigma_0, \sigma_1, \ldots, \sigma_n, \ldots \rangle$ belongs to the set $\mathcal{O}.F$ if and only if:

(1) $\mathcal{I}.F.\sigma_0$,
(2) $\forall i \in \mathbb{N} : (\mathcal{N}.F)^*.\{\sigma_{i+1}\}.\sigma_i$,
where $(\mathcal{N}.F)^*$ is the conjugate of $\mathcal{N}.F$ and $\{\sigma_{i+1}\}$ is the state predicate that evaluates to *true* for state σ_{i+1} and to *false* for any other state (i.e., $\lambda s.(s = \sigma_i)$).

The set $\mathcal{O}$ consists of those sequences of states that begin with an initial state that satisfies $\mathcal{I}$ and in which each state has a successor compatible with the transition function $\mathcal{N}$. This set is nonempty because $\mathcal{I}$ is satisfiable and $\mathcal{N}$ includes stuttering steps.

Once the computations of a transition system are built, next and inv specifications are defined as expected:

$$\begin{aligned} \mathsf{next}_{\mathcal{W}}.(p,q).F &\triangleq \forall \sigma \in \mathcal{O}.F : \forall i \in \mathbb{N} : p.\sigma_i \Rightarrow q.\sigma_{i+1}, \\ \mathsf{inv}_{\mathcal{W}}.p.F &\triangleq \forall \sigma \in \mathcal{O}.F : \forall i \in \mathbb{N} : p.\sigma_i. \end{aligned}$$

Informally, $\mathsf{inv}_{\mathcal{W}}.p$ means that any state of any computation of a system satisfies p. In the same way, $\mathsf{next}_{\mathcal{W}}.(p, q)$ means that, in any computation of the system, every state that satisfies p is immediately followed by a state that satisfies q. Although computations include stuttering steps, $\mathsf{next}_{\mathcal{W}}.(p, q)$ does *not* imply that $[p \Rightarrow q]$. It only implies $\mathsf{inv}_{\mathcal{W}}.(p \Rightarrow q)$.

Naturally, $\mathsf{next}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{W}}$ are related in a way similar to the relationship between $\mathsf{next}_{\mathcal{S}}$ and $\mathsf{inv}_{\mathcal{S}}$, namely

$$\mathsf{inv}_{\mathcal{W}}.p.F \;\equiv\; [\mathcal{I}.F \Rightarrow p] \;\wedge\; \mathsf{next}_{\mathcal{W}}.(p, p).F. \tag{$1_{\mathcal{W}}$}$$

In linear time temporal logic [6], $\mathsf{inv}_{\mathcal{W}}$ corresponds to $\Box$. In [2], "next" is called "co" and "next.(p, p)" is called "stable p".

2.4. Relationship between inductive and trace-based invariants

Although they are related, inductive and trace-based invariants are not equivalent. The use of the word *invariant*, all by itself, without making clear whether inductive or trace-based invariants are discussed, has been a source of great confusion. A famous example is the case of the UNITY formalism, as it was introduced in [1], in which $\mathsf{inv}_{\mathcal{S}}$ and $\mathsf{inv}_{\mathcal{W}}$ were used under the same name invariant. This led to inconsistencies that were heavily discussed at the time and solved in various ways, [7] being one of the earliest and cleanest solutions to the problem. Although the relationship between inductive and trace-based invariants is now well understood, the lack of an agreed upon terminology still makes it difficult to mention invariants without having to resort to stating definitions explicitly, as we have to do in this article. When stating a property that is valid for both types of invariant, we use the notation $\mathsf{inv}_{\{\mathcal{S},\mathcal{W}\}}$: If the occurrence of $\mathsf{inv}_{\{\mathcal{S},\mathcal{W}\}}$ in a proposition is replaced with $\mathsf{inv}_{\mathcal{S}}$ or $\mathsf{inv}_{\mathcal{W}}$, the resulting proposition is valid. We generalize this notation when $\mathsf{inv}_{\mathcal{E}}$ and $\mathsf{inv}_{\mathcal{U}}$ are introduced in Sections 5.2 and 5.3.

In this section, we examine the relationship between inductive and trace-based invariants in a context of closed systems. How these invariants differ is illustrated by an example in Section 2.5. Section 3 investigates the question of their relationship when composition is involved.

First, inductive invariants are *stronger* than trace-based invariants or, equivalently, trace-based invariants are a consequence of inductive invariants. For any system F and any state predicates p and q:

$$\begin{aligned} &\mathsf{next}_{\mathcal{S}}(p, q).F \;\Rightarrow\; \mathsf{next}_{\mathcal{W}}.(p, q).F, \\ &\quad \mathsf{inv}_{\mathcal{S}}.p.F \;\Rightarrow\; \mathsf{inv}_{\mathcal{W}}.p.F. \end{aligned} \tag{$2^{\mathcal{S}}_{\mathcal{W}}$}$$

Along with the following weakening rule (which does not hold for inductive invariants):

$$\mathsf{inv}_{\mathcal{W}}.(p \wedge q).F \Rightarrow \mathsf{inv}_{\mathcal{W}}.p.F,$$

it provides us with a technique for verifying trace-based invariants on a given transition system. To prove $\mathsf{inv}_{\mathcal{W}}.p.F$, one needs to find a state predicate r such that $\mathsf{inv}_{\mathcal{S}}.(p \wedge r).F$, which itself can be proved directly from $\mathcal{I}.F$ and $\mathcal{N}.F$. From a practical point of view, the difficulty lies in the discovery (or construction [9,10]) of predicate r. This technique is known in UNITY as the *substitution axiom* and in TLA as INV2, but one must keep in mind that the invariant being deduced is trace-based only.

This proof method is actually complete [2,7] in the sense that, when $\mathsf{inv}_{\mathcal{W}}.p.F$ holds, there is always a predicate r such that $\mathsf{inv}_{\mathcal{S}}.(p \wedge r).F$ is valid:

$$\mathsf{inv}_{\mathcal{W}}.p.F \;\equiv\; \exists r : \mathsf{inv}_{\mathcal{S}}.(p \wedge r).F. \tag{$3_{\mathcal{W}}$}$$

A similar relationship holds between $\mathsf{next}_{\mathcal{W}}$ and $\mathsf{next}_{\mathcal{S}}$:

$$\mathsf{next}_{\mathcal{W}}.(p, q).F \;\equiv\; \exists r : \mathsf{inv}_{\{\mathcal{S},\mathcal{W}\}}.r.F \;\wedge\; \mathsf{next}_{\mathcal{S}}.(p \wedge r, q).F. \tag{$4_{\mathcal{W}}$}$$

From the definition of $\mathsf{inv}_{\mathcal{S}}$ and the fact that $\mathcal{N}.F$ is universally conjunctive, $(3_{\mathcal{W}})$ can be reformulated as

$$\mathsf{inv}_{\mathcal{W}}.p.F \;\equiv\; \mathsf{inv}_{\mathcal{S}}.(p \wedge \forall r : \mathsf{inv}_{\mathcal{S}}.r.F : r).F.$$

$(\forall r : \mathsf{inv}_{\mathcal{S}}.r.F : r)$ is the conjunction of all those state predicates r that are inductively invariant in F. It is itself an inductive invariant of F (conjunctions of inductive invariants are inductive invariants), known as the *strongest invariant* of F [2,7]. We denote this state predicate by $\mathsf{SI}.F$:

$$[\mathsf{SI}.F \;\triangleq\; \forall r : \mathsf{inv}_{\mathcal{S}}.r.F : r]. \tag{$5_{\mathcal{S}}$}$$

Therefore, ($3_{\mathcal{W}}$) can still be rewritten as

$$\mathsf{inv}_{\mathcal{W}}.p.F \;\equiv\; \mathsf{inv}_{\mathcal{S}}.(p \wedge \mathsf{SI}.F). \tag{$6_{\mathcal{W}}$}$$

A similar relationship exists between $\mathsf{next}_{\mathcal{W}}$, $\mathsf{next}_{\mathcal{S}}$ and SI:

$$\mathsf{next}_{\mathcal{W}}.(p,q).F \;\equiv\; \mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SI}.F, q). \tag{$7_{\mathcal{W}}$}$$

The strongest invariant offers a characterization, convenient from a theoretical standpoint, of trace-based invariants:

$$\mathsf{inv}_{\mathcal{W}}.p.F \;\equiv\; [\mathsf{SI}.F \Rightarrow p]. \tag{$8_{\mathcal{W}}$}$$

In other words, p is satisfied by every state of every computation of F if and only if p is a consequence of $\mathsf{SI}.F$. Intuitively, this means that $\mathsf{SI}.F$ characterizes those states that can appear in F's computations, also known as the *reachable states* of F. It should be emphasized that we are referring here to those states that system F can reach *in isolation*, not if F is to become a component of a larger system.

As a final remark, we can observe that, because of ($8_{\mathcal{W}}$), $\mathsf{SI}.F$ is also the conjunction of all the trace-based invariants of F, and therefore

$$[\mathsf{SI}.F \;\equiv\; \forall r : \mathsf{inv}_{\{\mathcal{S},\mathcal{W}\}}.r.F : r].$$

In Section 5.2, we show how the strongest invariant can be redefined to take into account the interaction of a system with its environment, thus leading to a variety of specifications with good compositional properties that are not enjoyed by $\mathsf{inv}_{\mathcal{W}}$ and $\mathsf{next}_{\mathcal{W}}$.

2.5. Example

To illustrate the relationship between inductive and trace-based invariants, this section introduces a simple example of a system. This system is reused in Sections 3.3 and 5.4 when it becomes a component of a larger system, and in the final discussion.

At this point, we need a syntactic way of describing a transition system, particularly the transition $\mathcal{N}$. The syntax that we rely on in this paper is inspired by UNITY. It is by no means the best choice for representing transition systems. However, it is easy to read and should be directly accessible to a broad body of readers. TLA$^+$ was another possible choice, but because TLA's treatment of composition is quite different from ours (see the discussion in Section 7), this would have led to confusion.

Fig. 1 represents a transition system for a Door Manager. This system is responsible for opening the doors of an automatic train after the train has stopped, and for closing the doors before the train starts again. It can check the speed of the train but does not modify it. (The system might not open the doors at all, as we are not interested in progress properties here.)

$$\begin{array}{lcl}
\mathcal{V} & \triangleq & \texttt{doors}, \texttt{checkStop}, \texttt{speed} \\
\mathcal{L} & \triangleq & \texttt{doors}, \texttt{checkStop} \\
\Sigma & \triangleq & \texttt{doors}, \texttt{checkStop} \in \mathbb{B}; \texttt{speed} \in \mathbb{N} \\
\mathcal{I} & \triangleq & \neg\texttt{checkStop} \wedge \texttt{doors} = \texttt{closed} \\
\mathcal{N} & \triangleq & \texttt{checkStop} := (\texttt{speed} = 0) \\
 & | & \textsf{if } \texttt{checkStop} \textsf{ then } \texttt{doors} := \texttt{open} \\
 & | & \texttt{doors}, \texttt{checkStop} := \texttt{closed}, \texttt{false}
\end{array}$$

Fig. 1. System DoorManager.

$\mathcal{N}$ is represented by several state-changing statements separated by |. This operator | represents a nondeterministic choice in which a successor to the current state can be obtained by applying any one of these statements. It corresponds to [] in UNITY and $\vee$ in TLA. If a statement's guard is false, the next state is identical to the current state. Stuttering

transitions are not represented explicitly and the assignment operator := only modifies those state variables that appear on its left-hand side. The constants `open` and `closed` are aliases for *true* and *false*, respectively.

We consider the following invariant:

$$\mathsf{inv}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0). \tag{S}$$

Intuitively, this invariant says that, when the doors of a train are open, this train is stopped. If inv is $\mathsf{inv}_{\mathcal{S}}$, this specification is *not* satisfied by `DoorManager`: from a state where `checkStop` = *true*, `speed` = 1 and `doors` = `closed`, $\mathcal{N}$ may lead to a state where `speed` = 1 and `doors` = `open` (by choosing the second statement), thus falsifying the requirement that $[q \Rightarrow \mathcal{N}.\texttt{DoorManager}.q]$ for inductive invariants q.

No such state, however, can be reached by this system in isolation: it is impossible to have `checkStop` = *true* and `speed` = 1 at the same time in a reachable state. In other words, $[\mathsf{SI}.\texttt{DoorManager} \Rightarrow \neg(\texttt{checkStop} \wedge \texttt{speed} = 1)]$. Indeed, the following specification holds for this system:

$$\mathsf{inv}_{\mathcal{S}}.(\ (\texttt{checkStop} \;\Rightarrow\; \texttt{speed} = 0) \wedge (\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0)\ )$$

and, as a consequence of $(3_{\mathcal{W}})$, the following holds as well:

$$\mathsf{inv}_{\mathcal{W}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0).\texttt{DoorManager}.$$

So, the `DoorManager` system satisfies specification (S) when inv is $\mathsf{inv}_{\mathcal{W}}$ but not when inv is $\mathsf{inv}_{\mathcal{S}}$.

3. Composition of invariants

3.1. Parallel composition of transition systems

Parallel composition is defined here as interleaving of atomic actions, in a way similar to that in [1,2,5,11]: set-union of variables[1] and transitions, and conjunction of initial predicates. More precisely, if F and G are two transition systems, $F\|G$ is defined by:

- $\mathcal{V}.(F\|G) \triangleq \mathcal{V}.F \cup \mathcal{V}.G$.
- $\mathcal{L}.(F\|G) \triangleq \mathcal{L}.F \cup \mathcal{L}.G$.
- $\Sigma.(F\|G)$ maps variables from $\mathcal{V}.F$ to values as in $\Sigma.F$, and variables from $\mathcal{V}.G$ to values as in $\Sigma.G$.
- $\mathcal{I}.(F\|G) \triangleq \mathcal{I}.F \wedge \mathcal{I}.G$.
- $\mathcal{N}.(F\|G) \triangleq \mathcal{N}.F \wedge \mathcal{N}.G$.

Some parallel compositions, however, are impossible and do not result in transition systems, either because $\mathcal{N}.G$ (resp., $\mathcal{N}.F$) would conflict with $\mathcal{L}.F$ (resp., $\mathcal{L}.G$) (a component would modify another component's local variables) or because $\mathcal{I}.F$ and $\mathcal{I}.G$ are disjoint (no initial state is suitable for both components). We denote by $F\surd G$ the fact that systems F and G are compatible and can indeed be composed: $F\surd G$ is true exactly when:

(1) $\mathcal{I}.F \wedge \mathcal{I}.G$ is satisfiable, i.e., $\neg[\neg(\mathcal{I}.F \wedge \mathcal{I}.G)]$.
(2) For any state predicate q such that all free variables of q are in the set $\mathcal{L}.G$, $[q \Rightarrow \mathcal{N}.F.q]$; in other words, $\mathcal{N}.F$ does not modify variables from $\mathcal{L}.G$.
(3) For any state predicate q such that all free variables of q are in the set $\mathcal{L}.F$, $[q \Rightarrow \mathcal{N}.G.q]$.

It should be noted that, according to our definition, the local variables of a system can be read by other systems. A more standard definition would make a distinction between true local variables (which are invisible from outside the system) and "read-only" variables (which can be read but not written by other systems). As a consequence, $\surd$ allows for composite systems that would otherwise be ruled out. The rationale for our choice is that, although it is important not to allow these compositions from a system design point of view, it does not influence our formal treatment of composition of specifications. More precisely, there is no specification in our discussion that would be satisfied if external systems were not allowed to read local variables but is not satisfied if local variables can be read. Consequently, as regards the results presented in this article, true local variables and read-only variables are equivalent.

[1] We do not consider here the (important) issue of variable renaming.

3.2. Composition of inductive invariants

Inductive invariants are compositional in the following sense: If two systems F and G satisfy an inductive invariant $\mathsf{inv}_{\mathcal{S}}.p$, their parallel composition $F\|G$ satisfies this inductive invariant as well. This rule also holds for $\mathsf{next}_{\mathcal{S}}$ specifications. Formally, for any systems F and G such that $F\surd G$ holds and any state predicates p and q:

$$\mathsf{next}_{\mathcal{S}}.(p,q).F \wedge \mathsf{next}_{\mathcal{S}}.(p,q).G \;\Rightarrow\; \mathsf{next}_{\mathcal{S}}.(p,q).(F\|G), \tag{$9_{\mathcal{S}}$}$$

$$\mathsf{inv}_{\mathcal{S}}.p.F \wedge \mathsf{inv}_{\mathcal{S}}.p.G \;\Rightarrow\; \mathsf{inv}_{\mathcal{S}}.p.(F\|G). \tag{$10_{\mathcal{S}}$}$$

This is a direct consequence of the definition of inductive invariants and the definition of parallel composition of transition systems.

3.3. Composition of trace-based invariants

Trace-based invariants, on the other hand, do *not* always compose. There is no equivalent to ($9_{\mathcal{S}}$) and ($10_{\mathcal{S}}$) for $\mathsf{next}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{W}}$ because these specifications are defined in terms of $\mathcal{O}$, the set of computations of a closed system, and interactions with a possible environment are not taken into account at all in the definition of $\mathcal{O}$.

As an example, let us consider the Engine system of Fig. 2. This component is responsible for starting and stopping a train. In this simple implementation, a train can accelerate (and, in particular, begin to move) only when its doors are closed. It can decelerate at any time, as long as it is not already stopped. This system satisfies the following specification:

$$\mathsf{inv}_{\{\mathcal{S},\mathcal{W}\}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0).\texttt{Engine}.$$

If trace-based invariants were compositional, the system `DoorManager‖Engine` would satisfy $\mathsf{inv}_{\mathcal{W}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0)$, since both components satisfy it. Obviously, this is not the case, as `DoorManager‖Engine` admits computations in which a train starts to accelerate *after* variable `checkStop` is set to *true*, thus allowing the composed system to open the doors while the train is already in motion. So, `DoorManager‖Engine` illustrates the case of a system in which all components satisfy a desirable safety specification (a train must be halted for its doors to be open) but the system as a whole does not satisfy this specification.

4. Compositional specifications

4.1. Abstraction, reuse and information hiding

System designers expect component descriptions to be more concise and more abstract than component implementations. They want specifications to hide unnecessary implementation details and focus on the useful functionalities of a component. A primary reason for this is reuse. There is an overhead cost in specifying independent components, because a component is generally more difficult to verify than a closed system, including a closed system made of several components [12]. That cost can be balanced by the fact that a verified component is reused in building different systems, and therefore that the effort that went into its verification is reused as well [13,14].

$$\begin{array}{lcl}
\mathcal{V} & \triangleq & \texttt{doors}, \texttt{speed} \\
\mathcal{L} & \triangleq & \texttt{speed} \\
\Sigma & \triangleq & \texttt{doors} \in \mathbb{B};\ \texttt{speed} \in \mathbb{N} \\
\mathcal{I} & \triangleq & \texttt{speed} = 0 \\
\mathcal{N} & \triangleq & \textsf{if } \texttt{doors} = \texttt{closed} \textsf{ then } \texttt{speed} := \texttt{speed} + 1 \\
 & | & \textsf{if } \texttt{speed} > 0 \textsf{ then } \texttt{speed} := \texttt{speed} - 1
\end{array}$$

Fig. 2. System `Engine`.

Let us consider reusable components, first from the standpoint of a component provider, then from the standpoint of a component user. If a provider of a component F knows that F satisfies a useful property X, he or she would want X to be part of F's specification. This way, the fact that F satisfies X can be established once and for all. The alternative would be to provide users with a richer description of F (say, its implementation) and let them prove $X.F$ each time component F is used.

What if it turns out that what the users of F are really interested in is a property Y, weaker than X? If Y is used instead of X as the component specification, users of F will not need to prove $[X \Rightarrow Y]$ each time they use F as part of a larger system. As long as it can be successfully used for the verification of a composite system, specification Y is better than specification X.

This is even more true from the standpoint of a component user. If, in the course of building a system, a designer needs a Y component but is faced with a specification language that only allows for specification X to be expressed, this designer is left with a smaller set of components to choose from (there are more components that satisfy Y than there are that satisfy X). It may also be the case that specification X (what the designer is then required to describe) is difficult to construct from specification Y (what the designer actually needs).

As an illustration, let us consider the case of sequential composition of transformational programs. Let F be an iterative program (say, of the form "**while** G **do** B"), X its specification in terms of a loop invariant I, and Y its specification in terms of a precondition P and a postcondition Q. If F is specified in terms of X, $[X \Rightarrow Y]$ (in this case, $[P \Rightarrow I]$ and $[I \wedge \neg G \Rightarrow Q]$) must be proved every time the component is used. What is worse, designers looking for such a component need to provide the loop invariant of the component that they are looking for!

When dealing with sequential composition of transformational programs, there is no reason not to use a specification expressed as a pre/postcondition (P, Q) (other than the risk of choosing P to be too strong or Q to be too weak). This is because preconditions and postconditions can be used to reason about sequential composition. The situation that this article focuses on (parallel composition of reactive systems) is more problematic because we are faced with specifications that are suitable for reasoning about systems in isolation but are incompatible with concurrent composition. As discussed in the previous section, $\mathsf{inv}_{\mathcal{W}}$ is such a family of specifications: it emphasizes what is relevant (the reachable states) and hides unnecessary details (the inductive invariant used to verify it). The problem, however, is that $\mathsf{inv}_{\mathcal{W}}$ is *too* abstract and hides too much: the knowledge that all the components of a system satisfy $\mathsf{inv}_{\mathcal{W}}.p$ is not sufficient for asserting that the system itself satisfies $\mathsf{inv}_{\mathcal{W}}.p$.

As for inductive invariants $\mathsf{inv}_{\mathcal{S}}$, they are very much like the loop invariants mentioned above: it is impractical to expect designers to specify the components that they need in terms of their inductive invariants. Such invariants put too many constraints on the possible atomic transitions of a component, and call for a very specific implementation. For instance, if $\mathsf{inv}_{\mathcal{S}}$ is used in the train specification (S), the implementation from Fig. 2 becomes almost the only acceptable implementation of the engine, while in reality other implementations, like the program from Fig. 5, are perfectly suitable. We cannot expect system designers to have to provide such a big part of the implementation of the components that they are looking for.

Therefore, compositional reasoning requires a form of invariant that abstracts more from implementations than $\mathsf{inv}_{\mathcal{S}}$ but does not hide so much as to lose all compositional behavior, as $\mathsf{inv}_{\mathcal{W}}$ does. Two such forms of invariants are defined in this article (Sections 5.1 and 5.3). Their definition relies on a theory of specification transformers described in [14]. Elements from this theory are introduced in the remainder of this section. The reader is referred to [14] for more detail and for the proofs of the results used in this paper.

4.2. Existential and universal specifications

In this section and in the remainder of the paper, we rely on a terminology introduced in [15] to help make a distinction between specifications that compose according to rules similar to $(10_{\mathcal{S}})$ and those that follow different rules.

Consider formulas (UNIV) (similar to $(10_{\mathcal{S}})$) and (EXIST):

$$\mathsf{Spec}.F \wedge \mathsf{Spec}.G \Rightarrow \mathsf{Spec}.(F \| G) \qquad \text{(UNIV)}$$

$$\mathsf{Spec}.F \Rightarrow \mathsf{Spec}.(F \| G) \qquad \text{(EXIST)}$$

Specifications Spec that satisfy (UNIV) (for all systems F and G such that $F \surd G$) are called *universal*: a universal specification holds in any system in which all components satisfy the specification. Specifications that follow (EXIST)

are called *existential*: an existential specification holds in any system that contains at least one component that satisfies the specification. Of course, any existential specification is also universal.[2]

According to this terminology, the discussion in the previous section can be summarized as follows: (i) inductive invariants are universal, but they are not very abstract; (ii) trace-based invariants are abstract, but they are not universal in general. As mentioned earlier, it is the next part of the definition of invariants that is interesting with respect to composition. The part $[\mathcal{I}.F \Rightarrow p]$ is existential and does not present any difficulty (conjunctions of existential and universal specifications are universal).

4.3. Specification transformers

Many important component specifications are neither existential nor universal. A challenge for component and system designers is to derive compositional (existential or universal) specifications from non-compositional ones. To help study this fundamental question, we have defined a collection of predicate transformers that do indeed transform a given specification into a corresponding compositional specification [14]. In this paper, we rely on one of these transformers, namely WE. The reader is referred to [14] for properties of WE and related proofs.

It can be shown that for any specification Spec, there exists a (unique) weakest existential specification stronger than Spec, which we denote by WE.Spec. Formally, WE.Spec is the disjunction of all the existential specifications stronger than Spec:

$$\text{WE.Spec} \triangleq \exists X : \text{EXIST}(X) \wedge [X \Rightarrow \text{Spec}] : X$$

where $\text{EXIST}(X)$ means that specification X is existential:

$$\text{EXIST}(X) \triangleq \forall F, G : X.F \Rightarrow X.(F \| G)$$

WE.Spec characterizes those components that "bring" Spec into any system: if a component satisfies WE.Spec, then any system that uses this component will satisfy Spec. Conversely, if a component is such that any system that uses it satisfies Spec, this component satisfies WE.Spec. Given two specifications X and Y, $\text{WE}.(X \Rightarrow Y)$ can be used as a form of *assumption-commitment* specifications [14,15,17].

In this paper, we rely on the fact that predicate transformer WE is universally conjunctive. For any set $\mathcal{S}$ of specifications:

$$[\text{WE}.(\forall X : X \in \mathcal{S} : X) \equiv \forall X : X \in \mathcal{S} : \text{WE}.X].$$

Also, we use the fact that any universally conjunctive predicate transformer is monotonic, and hence that WE is monotonic:

$$[X \Rightarrow Y] \Rightarrow [\text{WE}.X \Rightarrow \text{WE}.Y].$$

The fact that WE.Spec is defined to be the *weakest* existential specification stronger than Spec relates to the discussion of the previous section regarding low-level and high-level specifications. Let us suppose that we are interested in designing a component so that systems that use this component satisfy specification Spec. If Spec is existential, we simply design a component that satisfies Spec. But if Spec is not existential, it is not enough for a component to satisfy Spec. The component must be designed to satisfy a stronger specification, namely WE.Spec. However, we only want to force on this component what is necessary to make systems that use this component satisfy Spec. In particular, we do not want this specification to include too many details about the component's implementation. We want those details to be hidden to a user of this component, and much freedom given to the implementer of such a component.

---

[2] For the sake of symmetry, (EXIST) should be written as $\text{Spec}.F \vee \text{Spec}.G \Rightarrow \text{Spec}.(F \| G)$. Because $\|$ is symmetric and F and G are universally quantified over, the two formulas are equivalent. However, we feel that (EXIST) better conveys the notion that systems "inherit" component properties when these properties are existential. It should be emphasized that (EXIST) is of a different form than $(9_{\mathcal{S}})$ or $(10_{\mathcal{S}})$. The rule for composing invariants in [16], translated into this paper's notation, is

$$\text{inv}.p.F \wedge \text{inv}.q.G \Rightarrow \text{inv}.(p \wedge q).(F \| G).$$

Although it has the flavor of rule $(10_{\mathcal{S}})$ for inductive invariants, it is actually a consequence of (EXIST) (which holds for invariants in [16]) and a simple rule about conjunctions of invariants being invariants. It is as if one feels that invariants *should* compose according to (UNIV), even in formalisms in which all specifications already compose according to (EXIST).

For some specifications Spec, there does not exist a weakest universal specification stronger than Spec. As a consequence, there is no transformer WU for universal composition that achieves what WE does for existential composition.

5. Compositional invariants

5.1. Existential invariants

Trace-based invariants do not compose existentially or universally. However, the transformer WE can be applied to $\mathsf{inv}_{\mathcal{W}}$ to obtain a form of invariant that composes existentially:

$$[\mathsf{inv}_{\mathcal{E}}.p \;\triangleq\; \mathsf{WE}.(\mathsf{inv}_{\mathcal{W}}.p)].$$

From the definition of WE, $\mathsf{inv}_{\mathcal{E}}$ is (existentially) compositional and stronger than $\mathsf{inv}_{\mathcal{W}}$. In other words, for any state predicate p and any systems F and G such that $F\surd G$:

$$\mathsf{inv}_{\mathcal{E}}.p.F \;\Rightarrow\; \mathsf{inv}_{\mathcal{E}}.p.(F\|G),$$
$$\mathsf{inv}_{\mathcal{E}}.p.F \;\Rightarrow\; \mathsf{inv}_{\mathcal{W}}.p.F.$$

Intuitively, existential invariants refer to the "private" part (local variables) of a component. Indeed, if $\mathsf{inv}_{\mathcal{E}}.p$ holds for component F, then predicate p only constrains the local variables of F and says nothing about the shared variables. Specifications $\mathsf{inv}_{\mathcal{E}}.p$ correspond to "cinvariant p" in Seuss [18], where *closures c* are defined in a way that is similar to our WE transformer.

5.2. Strongest invariant for composition

In Section 2.4, we defined the strongest invariant of a system to be the conjunction of all (inductive or trace-based) invariants of that system. In the same way, we define the *strongest invariant for composition* (SIC) of a system F to be the conjunction of all existential invariants of that system:

$$[\mathsf{SIC}.F \;\triangleq\; \forall r : \mathsf{inv}_{\mathcal{E}}.r.F : r]. \tag{$5_{\mathcal{E}}$}$$

Because conjunctions of invariants are invariants and the transformer WE is universally conjunctive, it follows that $\mathsf{SIC}.F$ is an existential invariant of F. As a consequence, $\mathsf{SIC}.F$ is also a trace-based invariant of F and $[\mathsf{SI}.F \Rightarrow \mathsf{SIC}.F]$. As it happens, $\mathsf{SIC}.F$ is actually an inductive invariant of F and

$$\mathsf{inv}_{\{\mathcal{E},\mathcal{S},\mathcal{W}\}}.(\mathsf{SIC}.F).F.$$

In the same way as the strongest invariant is used to characterize trace-based invariants using ($8_{\mathcal{W}}$), SIC can be used to characterize existential invariants. For any state predicate p and any system F,

$$\mathsf{inv}_{\mathcal{E}}.p.F \;\equiv\; [\mathsf{SIC}.F \Rightarrow p]. \tag{$8_{\mathcal{E}}$}$$

The following property can give a better understanding of SIC: $\mathsf{SIC}.F$ is the disjunction of all the predicates $\mathsf{SI}.(F\|G)$ for all systems G compatible with F:

$$[\mathsf{SIC}.F \;\equiv\; \exists G : F\surd G : \mathsf{SI}.(F\|G)].$$

Since SI characterizes the reachable states of a system in isolation, it follows that $\mathsf{SIC}.F$ characterizes those states that can be reached by system F *when F interacts with some compatible environment G*. Equivalently, states outside of $\mathsf{SIC}.F$ *cannot* be reached by systems in which F is a component (Fig. 3).

5.3. Universal invariants

Intuitively, if SIC replaces SI in formulas ($6_{\mathcal{W}}$) and ($7_{\mathcal{W}}$) for inv and next, the composition problem illustrated in Section 3.3 disappears: no new reachable state can interfere with the proof of an invariant when a system is composed. The following definitions of a new kind of next and inv specifications are based on this idea:

$$\mathsf{next}_{\mathcal{U}}.(p,q).F \;\triangleq\; \mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, q).F, \tag{$7_{\mathcal{U}}$}$$
$$\mathsf{inv}_{\mathcal{U}}.p.F \;\triangleq\; \mathsf{inv}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F).F. \tag{$6_{\mathcal{U}}$}$$

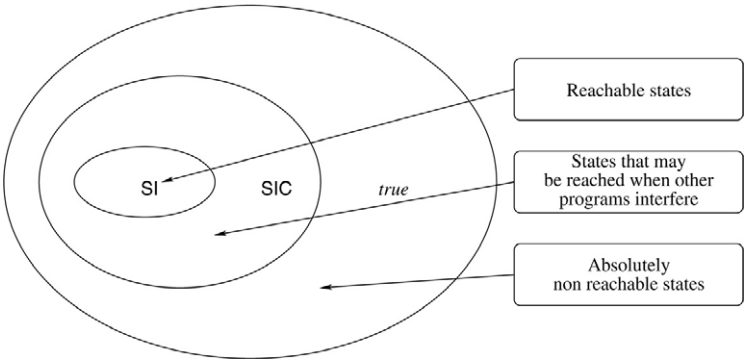

Fig. 3. State reachability.

Equivalently, $\mathsf{inv}_{\mathcal{U}}$ could have been defined in terms of $\mathsf{next}_{\mathcal{U}}$, in a way similar to $(1_{\mathcal{S}})$:

$$\mathsf{inv}_{\mathcal{U}}.p.F \;\equiv\; [\mathcal{I}.F \Rightarrow p] \;\wedge\; \mathsf{next}_{\mathcal{U}}.(p,p).F. \tag{$1_{\mathcal{U}}$}$$

Because SIC is defined to take the interaction of a system with its environment into account, contrary to SI, specifications $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$ enjoy compositional properties that $\mathsf{next}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{W}}$ do not share. More precisely:

Proposition 1. *For any state predicates p and q, $\mathsf{next}_{\mathcal{U}}.(p,q)$ and $\mathsf{inv}_{\mathcal{U}}.p$ are universal specifications:*

$$\mathsf{next}_{\mathcal{U}}.(p,q).F \wedge \mathsf{next}_{\mathcal{U}}.(p,q).G \;\Rightarrow\; \mathsf{next}_{\mathcal{U}}.(p,q).(F\|G), \tag{$9_{\mathcal{U}}$}$$

$$\mathsf{inv}_{\mathcal{U}}.p.F \wedge \mathsf{inv}_{\mathcal{U}}.p.G \;\Rightarrow\; \mathsf{inv}_{\mathcal{U}}.p.(F\|G). \tag{$10_{\mathcal{U}}$}$$

Proposition 1 follows from the fact that $\mathsf{SIC}.(F\|G)$ can be related to $\mathsf{SIC}.F$ and $\mathsf{SIC}.G$ in a simple way[3]:

$$[\mathsf{SIC}.(F\|G) \;\Rightarrow\; \mathsf{SIC}.F \wedge \mathsf{SIC}.G].$$

No such simple relationship exists between $\mathsf{SI}.(F\|G)$, $\mathsf{SI}.F$ and $\mathsf{SI}.G$. Intuitively, the formula means that any state reachable by a system in which $F\|G$ is a component is also reachable by a system in which F is a component (and a system in which G is a component), which is straightforward. It is not true of the reachable states of closed systems: a state can be reachable in $F\|G$ that was neither reachable in F nor in G.

Because SIC lies between SI and *true* (Fig. 3), $\mathsf{inv}_{\mathcal{U}}$ lies between $\mathsf{inv}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{S}}$. If a system F does not have any local variables, then $\mathsf{SIC}.F$ reduces to *true* (every state is reachable when F is composed because every variable of F can be written by its environment) and $\mathsf{inv}_{\mathcal{U}}$ is equivalent to $\mathsf{inv}_{\mathcal{S}}$. But when systems have local variables, $\mathsf{inv}_{\mathcal{U}}$ becomes strictly weaker than $\mathsf{inv}_{\mathcal{S}}$, while still being universal. This possibility, of course, is the interesting case: a component satisfies $\mathsf{inv}_{\mathcal{U}}.p$ (and, therefore, is suitable for composition) but does not satisfy $\mathsf{inv}_{\mathcal{S}}.p$ (and, therefore, would have to be ruled out if $\mathsf{inv}_{\mathcal{S}}$ were used in its specification instead of $\mathsf{inv}_{\mathcal{U}}$). This situation is illustrated in the following section.

We conclude this section with a discussion of how to prove $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$ specifications on a given system. Specifications $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$ can be verified without computing SIC explicitly, as $\mathsf{next}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{W}}$ do not require the explicit knowledge of SI. As with $\mathsf{inv}_{\mathcal{W}}$, the idea is to find a stronger predicate that is invariant inductively. But, for $\mathsf{inv}_{\mathcal{U}}$ to remain universal, this strengthening cannot be challenged by a component's environment (i.e., it needs to be an existential invariant).

Proposition 2. *For any system F and any predicates p and q:*

$$\mathsf{next}_{\mathcal{U}}.(p,q).F \;\equiv\; \exists r : \mathsf{inv}_{\mathcal{E}}.r.F \;\wedge\; \mathsf{next}_{\mathcal{S}}.(p\wedge r,q).F, \tag{$4_{\mathcal{U}}$}$$

$$\mathsf{inv}_{\mathcal{U}}.p.F \;\equiv\; \exists r : \mathsf{inv}_{\mathcal{E}}.r.F \;\wedge\; \mathsf{inv}_{\mathcal{S}}.(p\wedge r).F. \tag{$3_{\mathcal{U}}$}$$

Equivalences $(4_{\mathcal{U}})$ and $(3_{\mathcal{U}})$ are direct consequences of the definitions of SIC, $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$. They correspond to $(4_{\mathcal{W}})$ and $(3_{\mathcal{W}})$ for trace-based specifications. Because $\mathsf{inv}_{\mathcal{U}}.p$ is universal, it follows that a predicate p will be an

[3] Implication from right to left does not hold.

(abstract) invariant of a system if, for each component of that system, a "private" (i.e., existential) invariant can be found to strengthen p into an inductive invariant of this component.

Thanks to ($3_{\mathcal{U}}$) and ($4_{\mathcal{U}}$), all that is needed to prove $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$ specifications is a way to prove $\mathsf{inv}_{\mathcal{E}}.p$ on a transition system, which is achieved through the following proposition:

Proposition 3. *Given a system F and a state predicate p, $\mathsf{inv}_{\mathcal{E}}.p.F$ is true if there exists a state predicate r such that:*

(1) $\mathsf{inv}_{\mathcal{S}}.r.F$,
(2) $[r \Rightarrow p]$,
(3) *the free variables of r are a subset of $\mathcal{L}.F$.*

Another way of proving a specification $\mathsf{inv}_{\mathcal{E}}.p$ is expressed by the following proposition:

Proposition 4. *Given a system F and a state predicate p, $\mathsf{inv}_{\mathcal{E}}.p.F$ follows from:*

(1) $\mathsf{inv}_{\mathcal{U}}.p.F$,
(2) *the free variables of p are a subset of $\mathcal{L}.F$.*

An interesting use of Proposition 4 is illustrated in the example discussed in the following section.

5.4. Example

The component `DoorManager` from Fig. 1, which is incorrect, is now replaced with `DoorManager`′ from Fig. 4. This new component manages a set of lights in addition to opening and closing the doors of the train. More precisely, it turns the lights on as long as the doors are open and turns them off when the doors are closed. Values `on` and `off` are aliases for *true* and *false*, respectively.

$$
\begin{array}{lcl}
\mathcal{V} & \triangleq & \texttt{doors}, \texttt{speed}, \texttt{lights} \\
\mathcal{L} & \triangleq & \texttt{doors}, \texttt{lights} \\
\Sigma & \triangleq & \texttt{doors}, \texttt{lights} \in \mathbb{B}; \texttt{speed} \in \mathbb{N} \\
\mathcal{I} & \triangleq & \texttt{doors} = \texttt{closed} \wedge \texttt{lights} = \texttt{off} \\
\mathcal{N} & \triangleq & \mathsf{if}\ \texttt{speed} = 0 \vee \texttt{lights} = \texttt{on}\ \mathsf{then}\ \texttt{doors}, \texttt{lights} := \neg\texttt{doors}, \neg\texttt{lights}
\end{array}
$$

Fig. 4. System `DoorManager`′.

This component does *not* satisfy

$$\mathsf{inv}_{\mathcal{S}}.(\texttt{doors} = \texttt{open} \Rightarrow \texttt{speed} = 0).$$

However, it satisfies

$$\mathsf{inv}_{\mathcal{S}}.(\texttt{lights} = \texttt{on} \equiv \texttt{doors} = \texttt{open}),$$
$$\mathsf{inv}_{\mathcal{S}}.(\,(\texttt{lights} = \texttt{on} \equiv \texttt{doors} = \texttt{open}) \wedge (\texttt{doors} = \texttt{open} \Rightarrow \texttt{speed} = 0)\,).$$

Because `lights` and `doors` are local to `DoorManager`′, from Proposition 3 and ($3_{\mathcal{U}}$), it follows that

$$\mathsf{inv}_{\mathcal{U}}.(\texttt{doors} = \texttt{open} \Rightarrow \texttt{speed} = 0).\texttt{DoorManager}'.$$

Since this specification is also satisfied by `Engine` (it is satisfied as an inductive invariant), and it is universal from Proposition 1, it holds for the composite system `DoorManager`′‖`Engine` (and, therefore, the desired trace-based specification holds as well).

To continue with our train example, we consider the component `Engine`′ as described in Fig. 5. In this new implementation of the Engine, the speed does not necessarily increase or decrease by one, but varies according to an integer variable `acceleration`. As a consequence, this new component does not satisfy the inductive specification. However, it satisfies

$$\mathsf{inv}_{\mathcal{E}}.(\texttt{acceleration} > 0 \Rightarrow \texttt{speed} > 0).\texttt{Engine}'.$$

Using this existential invariant, we can then prove

$$\mathsf{inv}_{\mathcal{U}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0).\texttt{Engine}',$$

from which the correctness of the composed system can be deduced as before.

$$\begin{array}{lcl}
\mathcal{V} & \triangleq & \texttt{doors}, \texttt{speed}, \texttt{acceleration} \\
\mathcal{L} & \triangleq & \texttt{speed}, \texttt{acceleration} \\
\Sigma & \triangleq & \texttt{doors} \in \mathbb{B}; \texttt{speed}, \texttt{acceleration} \in \mathbb{N} \\
\mathcal{I} & \triangleq & \texttt{speed} = 0 \;\wedge\; \texttt{acceleration} = 0 \\
\mathcal{N} & \triangleq & \mathsf{if}\ \texttt{doors} = \texttt{closed}\ \mathsf{then}\ \texttt{speed} := 1 \\
 & | & \texttt{speed} := \texttt{speed} + \texttt{acceleration} \\
 & | & \mathsf{if}\ \texttt{speed} > \texttt{acceleration}\ \mathsf{then}\ \texttt{speed} := \texttt{speed} - \texttt{acceleration} \\
 & | & \mathsf{if}\ \texttt{speed} > 0\ \mathsf{then}\ \texttt{acceleration} := \texttt{acceleration} + 1 \\
 & | & \mathsf{if}\ \texttt{acceleration} > 0\ \mathsf{then}\ \texttt{acceleration} := \texttt{acceleration} - 1 \\
 & | & \mathsf{if}\ \texttt{speed} = 1\ \mathsf{then}\ \texttt{speed}, \texttt{acceleration} := 0, 0
\end{array}$$

Fig. 5. System $\texttt{Engine}'$.

Finally, we may need to compose $\texttt{DoorManager}'\|\texttt{Engine}'$ with the rest of the train system. Because variables doors and speed are now both local to $\texttt{DoorManager}'\|\texttt{Engine}'$, Proposition 4 can be used to deduce

$$\mathsf{inv}_{\mathcal{E}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0).(\texttt{DoorManager}'\|\texttt{Engine}').$$

Therefore, $\mathsf{inv}_{\{\mathcal{E},\mathcal{W}\}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0)$ will continue to hold when additional components are added to the system without any proof obligation on these components.

The train example illustrates three important points concerning $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$ specifications:

- When a component is specified in terms of $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$, there is a freedom in the way this component can be implemented. This freedom is not available when a component is specified in terms of $\mathsf{next}_{\mathcal{S}}$ and $\mathsf{inv}_{\mathcal{S}}$, because $\mathsf{next}_{\mathcal{S}}$ and $\mathsf{inv}_{\mathcal{S}}$ are less abstract than $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$.
- To prove a $\mathsf{next}_{\mathcal{U}}$ or an $\mathsf{inv}_{\mathcal{U}}$ specification on a given system is not different in nature from proving a $\mathsf{next}_{\mathcal{W}}$ or a $\mathsf{inv}_{\mathcal{W}}$ specification: it reduces to finding a suitable inductive invariant. The benefit is that, if some locality constraints hold for this inductive invariant, a universal specification can be claimed instead of a non-compositional specification.
- After some variables are made local, universal invariants can be promoted into existential invariants, hence simplifying subsequent composition steps.

6. Proofs related to SIC, $\mathsf{inv}_{\mathcal{E}}$, $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$

6.1. Preliminary standard results

In the following proofs, we use that fact, proved in [14], that WE is universally conjunctive. Moreover, it is clear from its expression in terms of SI ($8_{\mathcal{W}}$) that $\mathsf{inv}_{\mathcal{W}}$ is also universally conjunctive. It follows that $\mathsf{inv}_{\mathcal{E}}$ is universally conjunctive, as a composition of universally conjunctive transformers. As a consequence, all three functions (WE, $\mathsf{inv}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{E}}$) are monotonic.

Transformer $\mathsf{inv}_{\mathcal{W}}$ is *not* disjunctive. However, from its monotonicity, it follows that

$$[\mathsf{inv}_{\mathcal{W}}.(\exists p \in \mathcal{S} : p) \;\Leftarrow\; \exists p \in \mathcal{S} : \mathsf{inv}_{\mathcal{W}}.p] \tag{11}$$

Transformer $\mathcal{N}.F$ is universally conjunctive, and hence monotonic, from which the following property of $\mathsf{next}_{\mathcal{S}}$ is easily proved, for some domain $\mathcal{S}$:

$$(\forall x \in \mathcal{S} : \mathsf{next}_{\mathcal{S}}.(p(x), q(x))) \Rightarrow \mathsf{next}_{\mathcal{S}}.(\exists x \in \mathcal{S} : p(x), \exists x \in \mathcal{S} : q(x)) \tag{12}$$

A fundamental property of WE, proved in [14], is

$$\mathsf{WE}.\mathsf{Spec}.F \;\equiv\; \forall G : F\sqrt{G} : \mathsf{Spec}.(F\|G). \tag{13}$$

From the definition of $\mathsf{next}_{\mathcal{S}}$ and the fact that $\mathcal{N}.F$ is universally conjunctive, the following property of $\mathsf{next}_{\mathcal{S}}$ can be proved:

$$\mathsf{next}_{\mathcal{S}}.(p, q).F \wedge \mathsf{next}_{\mathcal{S}}.(p', q').F \;\Rightarrow\; \mathsf{next}_{\mathcal{S}}.(p \wedge p', q \wedge q').F \tag{14}$$

From the definition of $\mathsf{next}_{\mathcal{S}}$ and the definition of $\mathcal{N}.(F\|G)$, it follows that

$$\mathsf{next}_{\mathcal{S}}.(p, q).(F\|G) \;\Rightarrow\; \mathsf{next}_{\mathcal{S}}.(p, q).F \wedge \mathsf{next}_{\mathcal{S}}.(p, q).G. \tag{15}$$

(This shows that ($9_{\mathcal{S}}$) is actually an equivalence.)

6.2. $\mathit{inv}_{\mathcal{E}}.p.F \;\Rightarrow\; \mathit{inv}_{\mathcal{E}}.p.(F\|G)$

$\mathsf{inv}_{\mathcal{E}}.p$ is $\mathsf{WE}.(\mathsf{inv}_{\mathcal{W}}.p)$ and, from [14], $\mathsf{WE}.X$ is existential for any X.

6.3. $\mathit{inv}_{\mathcal{E}}.p.F \;\Rightarrow\; \mathit{inv}_{\mathcal{W}}.p.F$

$\mathsf{inv}_{\mathcal{E}}.p$ is $\mathsf{WE}.(\mathsf{inv}_{\mathcal{W}}.p)$ and, from [14], $\mathsf{WE}.X$ is stronger than X for any X.

6.4. $\mathit{inv}_{\{\mathcal{E},\mathcal{S},\mathcal{W}\}}.(\mathit{SIC}.F).F$

6.4.1. $\mathit{inv}_{\mathcal{E}}.(\mathit{SIC}.F).F$

Applying the fact that $\mathsf{inv}_{\mathcal{E}}$ is universally conjunctive to the definition of SIC, we have

$$[\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F) \;\equiv\; \forall r : \mathsf{inv}_{\mathcal{E}}.r.F : \mathsf{inv}_{\mathcal{E}}.r].$$

It follows that $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F$ is equivalent to $\forall r : \mathsf{inv}_{\mathcal{E}}.r.F : \mathsf{inv}_{\mathcal{E}}.r.F$, which is obviously true.

6.4.2. $\mathit{inv}_{\mathcal{W}}.(\mathit{SIC}.F).F$

This follows from $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F$, proved above, by weakening.

6.4.3. $\mathit{inv}_{\mathcal{S}}.(\mathit{SIC}.F).F$

See proof in Section 6.8.

6.5. $\mathit{inv}_{\mathcal{E}}.p.F \;\equiv\; [\mathit{SIC}.F \Rightarrow p]$ ($8_{\mathcal{E}}$)

From left to right: assume $\mathsf{inv}_{\mathcal{E}}.p.F$. Then,

	$\mathsf{SIC}.F$	
$=$	$\forall q : \mathsf{inv}_{\mathcal{E}}.q.F : q$	from the definition of SIC
$\Rightarrow$	$\mathsf{inv}_{\mathcal{E}}.p.F \Rightarrow p$	by choosing $q = p$
$=$	p	from the assumption $\mathsf{inv}_{\mathcal{E}}.p.F$

From right to left,

	$[\mathsf{SIC}.F \Rightarrow p]$	
$\Rightarrow$	$[\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F) \Rightarrow \mathsf{inv}_{\mathcal{E}}.p]$	from the monotonicity of $\mathsf{inv}_{\mathcal{E}}$
$\Rightarrow$	$\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F \Rightarrow \mathsf{inv}_{\mathcal{E}}.p.F$	from the definition of the everywhere operator
$=$	$\mathsf{inv}_{\mathcal{E}}.p.F$	from $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F$, proved in 6.4.1

6.6. $[\mathit{SIC}.F \equiv \exists G : F\surd G : \mathit{SI}.(F\|G)]$

From left to right,

$$
\begin{array}{lll}
 & [\mathsf{SIC}.F \Rightarrow \exists G : F\surd G : \mathsf{SI}.(F\|G)] & \\
= & \mathsf{inv}_{\mathcal{E}}.(\exists G : F\surd G : \mathsf{SI}.(F\|G)).F & \text{from } \mathsf{inv}_{\mathcal{E}}.p.F \equiv [\mathsf{SIC}.F \Rightarrow p], \text{ proved in 6.5} \\
= & \mathsf{WE}.(\mathsf{inv}_{\mathcal{W}}.(\exists G : F\surd G : \mathsf{SI}.(F\|G))).F & \text{by definition of } \mathsf{inv}_{\mathcal{E}} \\
= & \forall H : F\surd H : & \\
 & \quad \mathsf{inv}_{\mathcal{W}}.(\exists G : F\surd G : \mathsf{SI}.(F\|G)).(F\|H) & \text{from property (13) of WE} \\
\Leftarrow & \forall H : F\surd H : & \text{from property (11) of } \mathsf{inv}_{\mathcal{W}} \text{ and the monotonicity of } \forall \\
 & \quad \exists G : F\surd G : \mathsf{inv}_{\mathcal{W}}.(\mathsf{SI}.(F\|G)).(F\|H) & \\
\Leftarrow & \forall H : F\surd H : & \text{by choosing } G = H \text{ and the monotonicity of } \forall \\
 & \quad (F\surd H) \wedge \mathsf{inv}_{\mathcal{W}}.(\mathsf{SI}.(F\|H)).(F\|H) & \\
= & \forall H : F\surd H : F\surd H & \text{from the relationship between SI and } \mathsf{inv}_{\mathcal{W}} \ (8_{\mathcal{W}}) \\
= & \mathit{true} & \text{obviously}
\end{array}
$$

From right to left,

$$
\begin{array}{lll}
 & \exists G : F\surd G : \mathsf{SI}.(F\|G) & \\
= & \exists G : F\surd G : (\forall r : \mathsf{inv}_{\mathcal{W}}.r.(F\|G) : r) & \text{from property } [\mathsf{SI}.F \equiv \forall r : \mathsf{inv}_{\{\mathcal{S},\mathcal{W}\}}.r.F \Rightarrow r] \\
\Rightarrow & \forall r : (\exists G : F\surd G : \mathsf{inv}_{\mathcal{W}}.r.(F\|G) \Rightarrow r) & \text{by swapping quantifiers: } \exists\forall \Rightarrow \forall\exists \\
= & \forall r \ : \ (\exists G : F\surd G : \neg\mathsf{inv}_{\mathcal{W}}.r.(F\|G)) & \text{by distributing } \exists \text{ over } \vee \\
 & \quad \vee \ (\exists G : F\surd G : r) & \\
= & \forall r \ : \ (\exists G : F\surd G : \neg\mathsf{inv}_{\mathcal{W}}.r.(F\|G)) & \text{no free } G \text{ in } r \\
 & \quad \vee \ ((\exists G : F\surd G) \wedge r) & \\
\Rightarrow & \forall r : (\exists G : F\surd G : \neg\mathsf{inv}_{\mathcal{W}}.r.(F\|G)) \vee r & \text{from the monotonicity of } \forall, \vee \\
= & \forall r : \neg(\forall G : F\surd G : \mathsf{inv}_{\mathcal{W}}.r.(F\|G)) \vee r & \text{by De Morgan} \\
= & \forall r : (\forall G : F\surd G : \mathsf{inv}_{\mathcal{W}}.r.(F\|G)) : r & \text{from predicate calculus} \\
= & \forall r : \mathsf{WE}.(\mathsf{inv}_{\mathcal{W}}.r).F : r & \text{from property (13) of WE} \\
= & \forall r : \mathsf{inv}_{\mathcal{E}}.r.F : r & \text{from the definition of } \mathsf{inv}_{\mathcal{E}} \\
= & \mathsf{SIC}.F & \text{from the definition of SIC}
\end{array}
$$

6.7. $[\mathit{SIC}.(F\|G) \Rightarrow \mathit{SIC}.F \wedge \mathit{SIC}.G]$

We prove that $[\mathsf{SIC}.(F\|G) \Rightarrow \mathsf{SIC}.F]$. The proof for G is identical:

$$
\begin{array}{lll}
 & \mathsf{SIC}.(F\|G) & \\
= & \forall r : \mathsf{inv}_{\mathcal{E}}.r.(F\|G) : r & \text{from the definition of SIC} \\
\Rightarrow & \forall r : \mathsf{inv}_{\mathcal{E}}.r.F : r & \text{because } \mathsf{inv}_{\mathcal{E}}.r \text{ is existential, proved in 6.2, and } \forall \text{ is antimonotonic in its range} \\
= & \mathsf{SIC}.F & \text{from the definition of SIC}
\end{array}
$$

6.8. $\mathit{inv}_{\mathcal{S}}.(\mathit{SIC}.F).F$

By definition of $\mathsf{inv}_{\mathcal{S}}$ $(1_{\mathcal{S}})$, $\mathsf{inv}_{\mathcal{S}}.(\mathsf{SIC}.F).F$ is equivalent to $[\mathcal{I}.F \Rightarrow \mathsf{SIC}.F] \wedge \mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F, \mathsf{SIC}.F).F$. The first conjunct follows easily from $\mathsf{inv}_{\mathcal{W}}.(\mathsf{SIC}.F).F$, proved above, and $(6_{\mathcal{W}})$. We only prove the second conjunct.

As already mentioned, the strongest invariant of a system is a strong invariant of this system. Therefore,

$$
\begin{array}{ll}
 & \forall G : F \surd G : \mathsf{inv}_{\mathcal{S}}.(\mathsf{SI}.(F\|G)).(F\|G) \\
\Rightarrow & \forall G : F \surd G : \mathsf{next}_{\mathcal{S}}.(\mathsf{SI}.(F\|G), \mathsf{SI}.(F\|G)).(F\|G) \text{ from the definition of } \mathsf{inv}_{\mathcal{S}}\ (1_{\mathcal{S}}) \\
\Rightarrow & \forall G : F \surd G : \mathsf{next}_{\mathcal{S}}.(\mathsf{SI}.(F\|G), \mathsf{SI}.(F\|G)).F \text{ from (15) and the monotonicity of } \forall \\
\Rightarrow & \mathsf{next}_{\mathcal{S}}.(\exists G : F \surd G : \mathsf{SI}.(F\|G), \exists G : F \surd G : \mathsf{SI}.(F\|G)).F \text{ from (12)} \\
= & \mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F, \mathsf{SIC}.F).F \text{ from } [\mathsf{SIC}.F \equiv \exists G : F \surd G : \mathsf{SI}.(F\|G)], \text{ proved in 6.6}
\end{array}
$$

6.9. $\mathsf{inv}_{\mathcal{U}}.p.F \equiv [\mathcal{I}.F \Rightarrow p] \wedge \mathsf{next}_{\mathcal{U}}.(p, p).F$ $(1_{\mathcal{U}})$

From left to right,

1.	$\mathsf{inv}_{\mathcal{U}}.p.F$	by assumption
2.	$\mathsf{inv}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F).F$	from line 1 and the definition of $\mathsf{inv}_{\mathcal{U}}$ $(6_{\mathcal{U}})$
3.	$\mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, p \wedge \mathsf{SIC}.F).F \wedge [\mathcal{I}.F \Rightarrow p \wedge \mathsf{SIC}.F]$	from line 2 and the definition of $\mathsf{inv}_{\mathcal{S}}$ $(1_{\mathcal{S}})$
4.	$[\mathcal{I}.F \Rightarrow p]$	from line 3 by weakening
5.	$[p \wedge \mathsf{SIC}.F \Rightarrow \mathcal{N}.F.(p \wedge \mathsf{SIC}.F)]$	from line 3 and the definition of $\mathsf{next}_{\mathcal{S}}$
6.	$[p \wedge \mathsf{SIC}.F \Rightarrow \mathcal{N}.F.p \wedge \mathcal{N}.F.(\mathsf{SIC}.F)]$	from line 5 because $\mathcal{N}.F$ in conjunctive
7.	$[p \wedge \mathsf{SIC}.F \Rightarrow \mathcal{N}.F.p]$	from line 6 by weakening
8.	$\mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, p).F$	from line 7 and the definition of $\mathsf{next}_{\mathcal{S}}$
9.	$\mathsf{next}_{\mathcal{U}}.(p, p).F$	from line 8 and the definition of $\mathsf{next}_{\mathcal{U}}$ $(7_{\mathcal{U}})$
10.	$[\mathcal{I}.F \Rightarrow p] \wedge \mathsf{next}_{\mathcal{U}}.(p, p).F$	from lines 4 and 9

From right to left,

1.	$[\mathcal{I}.F \Rightarrow p]$	by assumption
2.	$\mathsf{next}_{\mathcal{U}}.(p, p).F$	by assumption
3.	$\mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, p).F$	from line 2 and the definitionof $\mathsf{next}_{\mathcal{U}}$ $(7_{\mathcal{U}})$
4.	$\mathsf{inv}_{\mathcal{S}}.(\mathsf{SIC}.F).F$	proved in 6.8
5.	$\mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F, \mathsf{SIC}.F).F$	from line 4 and the definition of $\mathsf{inv}_{\mathcal{S}}$ $(1_{\mathcal{S}})$
6.	$\mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, p \wedge \mathsf{SIC}.F).F$	from lines 3 and 5 using (14)
7.	$[\mathcal{I}.F \Rightarrow \mathsf{SIC}.F]$	from line 4 and the definition of $\mathsf{inv}_{\mathcal{S}}$ $(1_{\mathcal{S}})$
8.	$[\mathcal{I}.F \Rightarrow p \wedge \mathsf{SIC}.F]$	from lines 1 and 7
9.	$\mathsf{inv}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F).F$	from lines 6 and 8 and the definition of $\mathsf{inv}_{\mathcal{S}}$ $(1_{\mathcal{S}})$
10.	$\mathsf{inv}_{\mathcal{U}}.p.F$	from line 9 and the definition of $\mathsf{inv}_{\mathcal{U}}$ $(6_{\mathcal{U}})$

6.10. $\mathsf{next}_{\mathcal{U}}.(p, q)$ *and* $\mathsf{inv}_{\mathcal{U}}.p$ *are universal specifications* *(Proposition 1)*

6.10.1. $\mathsf{next}_{\mathcal{U}}.(p, q)$ *is universal* $(9_{\mathcal{U}})$

Given two systems F and G such that $F \surd G$, and two state predicates p and q such that $\mathsf{next}_{\mathcal{U}}.(p, q).F \wedge \mathsf{next}_{\mathcal{U}}.(p, q).G$, we prove $\mathsf{next}_{\mathcal{U}}.(p, q).(F\|G)$:

	$\mathsf{next}_{\mathcal{U}}.(p,q).F \wedge \mathsf{next}_{\mathcal{U}}.(p,q).G$	by assumption
$=$	$\quad \mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, q).F$	from the definition of $\mathsf{next}_{\mathcal{U}}$ ($7_{\mathcal{U}}$)
	$\wedge \quad \mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.G, q).G$	
$=$	$\quad (p \wedge \mathsf{SIC}.F \Rightarrow \mathcal{N}.F.q)$	from the definition of $\mathsf{next}_{\mathcal{S}}$
	$\wedge \quad (p \wedge \mathsf{SIC}.G \Rightarrow \mathcal{N}.G.q)$	
$\Rightarrow$	$p \wedge \mathsf{SIC}.F \wedge \mathsf{SIC}.G \Rightarrow \mathcal{N}.F.q \wedge \mathcal{N}.G.q$	from predicate calculus
$=$	$p \wedge \mathsf{SIC}.F \wedge \mathsf{SIC}.G \Rightarrow \mathcal{N}.(F\|G).q$	from the definition of $\mathcal{N}.(F\|G)$
$\Rightarrow$	$p \wedge \mathsf{SIC}.(F\|G) \Rightarrow \mathcal{N}.(F\|G).q$	from property of SIC, proved in 6.7
$=$	$\mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.(F\|G), q).(F\|G)$	from the definition of $\mathsf{next}_{\mathcal{S}}$
$=$	$\mathsf{next}_{\mathcal{U}}.(p,q).(F\|G)$	from the definition of $\mathsf{next}_{\mathcal{U}}$ ($7_{\mathcal{U}}$)

6.10.2. $\mathsf{inv}_{\mathcal{U}}.p$ is universal ($10_{\mathcal{U}}$)

Given two systems F and G such that $F\sqrt{G}$, and a state predicate p such that $\mathsf{inv}_{\mathcal{U}}.p.F \wedge \mathsf{inv}_{\mathcal{U}}.p.G$, we prove $\mathsf{inv}_{\mathcal{U}}.p.(F\|G)$. On rewriting both assumptions and the goal according to $\mathsf{inv}_{\mathcal{U}}.p \equiv [\mathcal{I} \Rightarrow p] \wedge \mathsf{next}_{\mathcal{U}}.(p,p)$, proved in 6.9, the proof obligation becomes:

- $\mathsf{next}_{\mathcal{U}}.(p,p).F \wedge \mathsf{next}_{\mathcal{U}}.(p,p).G \Rightarrow \mathsf{next}_{\mathcal{U}}.(p,p).(F\|G)$
- $[\mathcal{I}.F \Rightarrow p] \wedge [\mathcal{I}.G \Rightarrow p] \Rightarrow [\mathcal{I}.(F\|G) \Rightarrow p]$

The first item follows from the fact that $\mathsf{next}_{\mathcal{U}}$ is universal, proved above. The second item follows easily from the definition of $\mathcal{I}.(F\|G)$.

6.11. $\mathsf{next}_{\mathcal{U}}.(p,q).F \equiv \exists r : \mathsf{inv}_{\mathcal{E}}.r.F \wedge \mathsf{next}_{\mathcal{S}}.(p \wedge r, q).F$ ($4_{\mathcal{U}}$)

From left to right: choose $\mathsf{SIC}.F$ for r and use the fact that $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F$, proved in 6.4.1.
From right to left: choose r such that $\mathsf{inv}_{\mathcal{E}}.r.F$ and $\mathsf{next}_{\mathcal{S}}.(p \wedge r, q).F$. Then,

1.	$[p \wedge r \Rightarrow \mathcal{N}.F.q]$	by definition of $\mathsf{next}_{\mathcal{S}}$
2.	$[p \wedge r \wedge \mathsf{SIC}.F \Rightarrow \mathcal{N}.F.q]$	from line 1 by strengthening
3.	$[\mathsf{SIC}.F \Rightarrow r]$	from $\mathsf{inv}_{\mathcal{E}}.r.F$, using a property proved in 6.5
4.	$[p \wedge \mathsf{SIC}.F \Rightarrow \mathcal{N}.F.q]$	from lines 2 and 3 above, by absorption
5.	$\mathsf{next}_{\mathcal{S}}.(p \wedge \mathsf{SIC}.F, q).F$	from line 4 by definition of $\mathsf{next}_{\mathcal{S}}$
6.	$\mathsf{next}_{\mathcal{U}}.(p,q).F$	from line 6 by definition of $\mathsf{next}_{\mathcal{U}}$ ($7_{\mathcal{U}}$)

6.12. $\mathsf{inv}_{\mathcal{U}}.p.F \equiv \exists r : \mathsf{inv}_{\mathcal{E}}.r.F \wedge \mathsf{inv}_{\mathcal{S}}.(p \wedge r).F$ ($3_{\mathcal{U}}$)

From left to right: choose $\mathsf{SIC}.F$ for r and use the fact that $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F$, proved in 6.4.1.
From right to left: choose r such that $\mathsf{inv}_{\mathcal{E}}.r.F$ and $\mathsf{inv}_{\mathcal{S}}.(p \wedge r).F$. Then,

1.	$\mathsf{inv}_{\mathcal{S}}.(\mathsf{SIC}.F).F$	proved in 6.8
2.	$\mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F, \mathsf{SIC}.F) \wedge [\mathcal{I}.F \Rightarrow \mathsf{SIC}.F]$	from line 1 and the definition of $\mathsf{inv}_{\mathcal{S}}$ ($1_{\mathcal{S}}$)
3.	$\mathsf{next}_{\mathcal{S}}.(p \wedge r, p \wedge r) \wedge [\mathcal{I}.F \Rightarrow p \wedge r]$	from the definition of $\mathsf{inv}_{\mathcal{S}}$ ($1_{\mathcal{S}}$)
4.	$\mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F, \mathsf{SIC}.F) \wedge \mathsf{next}_{\mathcal{S}}.(p \wedge r, p \wedge r)$ $\wedge [\mathcal{I}.F \Rightarrow \mathsf{SIC}.F \wedge p \wedge r]$	from lines 2 and 3 above
5.	$\mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F \wedge p \wedge r, \mathsf{SIC}.F \wedge p \wedge r).F$ $\wedge [\mathcal{I}.F \Rightarrow \mathsf{SIC}.F \wedge p \wedge r]$	from line 4 using (14)

6.	$[\mathsf{SIC}.F \Rightarrow r]$	from assumption $\mathsf{inv}_{\mathcal{E}}.r.F$, using property proved in 6.5
7.	$\mathsf{next}_{\mathcal{S}}.(\mathsf{SIC}.F \wedge p, \mathsf{SIC}.F \wedge p).F$ $\wedge\, [\mathcal{I}.F \Rightarrow \mathsf{SIC}.F \wedge p]$	from lines 5 and 6 above
8.	$\mathsf{inv}_{\mathcal{S}}.(\mathsf{SIC}.F \wedge p).F$	from line 7 and the definition of $\mathsf{inv}_{\mathcal{S}}$ ($1_{\mathcal{S}}$)
9.	$\mathsf{inv}_{\mathcal{U}}.p.F$	from line 8 and the definition of $\mathsf{inv}_{\mathcal{U}}$ ($6_{\mathcal{U}}$)

6.13. Propositions 3 and 4

We first prove the following, more general proposition:

Proposition 5. *Given a system F and state predicates p and q, $\mathsf{inv}_{\mathcal{E}}.(p \wedge q).F$ follows from:*

(1) $\mathsf{inv}_{\mathcal{E}}.p.F$,
(2) $\mathsf{inv}_{\mathcal{S}}.(p \wedge q).F$,
(3) *the free variables of q are a subset of $\mathcal{L}.F$.*

To prove Proposition 5, we assume a system F and state predicates p and q such that $\mathsf{inv}_{\mathcal{E}}.p.F$, $\mathsf{inv}_{\mathcal{S}}.(p \wedge q).F$, and the free variables of q are a subset of $\mathcal{L}.F$.

$\mathsf{inv}_{\mathcal{E}}.(p \wedge q).F$ follows from $\mathsf{inv}_{\mathcal{E}}.q.F$ and the assumption $\mathsf{inv}_{\mathcal{E}}.p.F$, using the fact that $\mathsf{inv}_{\mathcal{E}}$ is conjunctive. To prove $\mathsf{inv}_{\mathcal{E}}.q.F$, we consider a system G such that $F \surd G$:

1.	$[\mathcal{I}.(F \| G) \Rightarrow q]$	from the assumptions and the definition of $\mathcal{I}.(F \| G)$
2.	$\mathsf{next}_{\mathcal{S}}.(p \wedge q, p \wedge q).F$	from $\mathsf{inv}_{\mathcal{S}}.(p \wedge q).F$ and the definition of $\mathsf{inv}_{\mathcal{S}}$ ($1_{\mathcal{S}}$)
3.	$[p \wedge q \Rightarrow \mathcal{N}.F.(p \wedge q)]$	from line 2 and the definition of $\mathsf{next}_{\mathcal{S}}$
4.	$[p \wedge q \Rightarrow \mathcal{N}.F.p \wedge \mathcal{N}.F.q]$	from line 3 because $\mathcal{N}.F$ is conjunctive
5.	$[p \wedge q \Rightarrow \mathcal{N}.F.q]$	from line 4 by weakening
6.	$\mathsf{next}_{\mathcal{S}}.(q, q).G$	from the assumption on the free variables of q, the assumption $F \surd G$ and the definition of $\surd$
7.	$[q \Rightarrow \mathcal{N}.G.q]$	from line 6 and the definition of $\mathsf{next}_{\mathcal{S}}$
8.	$[p \wedge q \Rightarrow \mathcal{N}.G.q]$	from line 7 by strengthening
9.	$[p \wedge q \Rightarrow \mathcal{N}.F.q \wedge \mathcal{N}.G.q]$	from lines 5 and 8 above
10.	$[p \wedge q \Rightarrow \mathcal{N}.(F \| G).q]$	from line 9 and the definition of $\mathcal{N}.(F \| G)$
11.	$\mathsf{next}_{\mathcal{S}}.(p \wedge q, q).(F \| G)$	from line 10 and the definition of $\mathsf{next}_{\mathcal{S}}$
12.	$\mathsf{next}_{\mathcal{W}}.(p \wedge q, q).(F \| G)$	from line 11 by weakening
13.	$\mathsf{inv}_{\mathcal{E}}.p.(F \| G)$	from assumption $\mathsf{inv}_{\mathcal{E}}.p.F$ and the fact that $\mathsf{inv}_{\mathcal{E}}.p$ is existential
14.	$\mathsf{inv}_{\mathcal{W}}.p.(F \| G)$	from line 13 by weakening
15.	$[\mathsf{SI}.(F \| G) \Rightarrow p]$	from line 14 using ($8_{\mathcal{W}}$)
16.	$\mathsf{next}_{\mathcal{S}}.(p \wedge q \wedge \mathsf{SI}.(F \| G), q).(F \| G)$	from line 12 using ($7_{\mathcal{W}}$)
17.	$\mathsf{next}_{\mathcal{S}}.(q \wedge \mathsf{SI}.(F \| G), q).(F \| G)$	from lines 15 and 16 above
18.	$\mathsf{next}_{\mathcal{W}}.(q, q).(F \| G)$	from line 17 using ($7_{\mathcal{W}}$)
19.	$\mathsf{inv}_{\mathcal{W}}.q.(F \| G)$	from lines 1 and 18
20.	$\forall G : (F \surd G) \Rightarrow \mathsf{inv}_{\mathcal{W}}.q.(F \| G)$	from line 19 by generalization
21.	$\mathsf{WE}.(\mathsf{inv}_{\mathcal{W}}.q).F$	from line 20 and property (13) of WE
22.	$\mathsf{inv}_{\mathcal{E}}.q.F$	from line 21 and the definition of $\mathsf{inv}_{\mathcal{E}}$

6.13.1. Proposition 3

We choose a system F and a state predicate r such that $\mathsf{inv}_{\mathcal{S}}.r.F$, $[r \Rightarrow p]$ and the free variables of r are a subset of $\mathcal{L}.F$. We apply Proposition 5 with $p := (r \Rightarrow p)$ and $q := r$:

- $\mathsf{inv}_{\mathcal{E}}.(r \Rightarrow p).F$ holds because $[(r \Rightarrow p) \equiv \mathit{true}]$,
- $\mathsf{inv}_{\mathcal{S}}.(r \wedge (r \Rightarrow p)).F$ holds because $[r \wedge (r \Rightarrow p) \equiv r]$ and $\mathsf{inv}_{\mathcal{S}}.r.F$ holds,
- the free variables of q are a subset of $\mathcal{L}.F$ because q is r.

From Proposition 5, we deduce $\mathsf{inv}_{\mathcal{E}}.(r \wedge (r \Rightarrow p)).F$, which is equivalent to $\mathsf{inv}_{\mathcal{E}}.(r \wedge p).F$, from which $\mathsf{inv}_{\mathcal{E}}.p.F$ follows by the monotonicity of $\mathsf{inv}_{\mathcal{E}}$.

6.13.2. Proposition 4

We choose a system F and a state predicate p such that $\mathsf{inv}_{\mathcal{U}}.p.F$ and the free variables of p are a subset of $\mathcal{L}.F$. We apply Proposition 5 with $p := \mathsf{SIC}.F$ and $q := p$:

- $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F).F$ holds from 6.4.1,
- $\mathsf{inv}_{\mathcal{S}}.(\mathsf{SIC}.F \wedge p).F$ holds from $\mathsf{inv}_{\mathcal{U}}.p.F$ and the definition of $\mathsf{inv}_{\mathcal{U}}$,
- the free variables of q are a subset of $\mathcal{L}.F$ because q is p.

From Proposition 5, we get $\mathsf{inv}_{\mathcal{E}}.(\mathsf{SIC}.F \wedge p).F$, from which $\mathsf{inv}_{\mathcal{E}}.p.F$ follows by the monotonicity of $\mathsf{inv}_{\mathcal{E}}$.

7. Summary and discussion

Compositional reasoning has been defined as the capacity to deduce the correctness of a system from the specifications (as opposed to implementations) of its components [19]. Although it can be argued that there is more to compositional reasoning than just this bottom-up deduction [14,20], this question is indeed an important issue that must be dealt with. However, for the problem to be fully defined, one needs to add the important additional constraint, which too often is left implicit, that we expect specifications to be more abstract than implementations. Without this constraint, the problem could be solved easily by choosing a low-level specification language close to implementation languages. After all, CSP processes [21], UNITY programs and TLA formulas are formally defined and compose very naturally. The real issue, however, is defining abstract, high-level specifications such that the behavior of a system can be analyzed from the specifications of its components. Only in this case is compositional reasoning worthwhile, because substantial verification efforts are embedded in components and *reused* when components are reused.

Of the two most classic forms of invariants, $\mathsf{inv}_{\mathcal{W}}$ is too abstract for composition and does not allow for system invariants to be deduced from component invariants. On the other hand, $\mathsf{inv}_{\mathcal{S}}$ is not abstract enough and, although it is compositional, it reveals too much of a component (local state and atomic transitions) and is not well suited to the specification of reusable components. In this article, we define two new forms of invariants, $\mathsf{inv}_{\mathcal{E}}$ and $\mathsf{inv}_{\mathcal{U}}$. Both are compositional, albeit in different ways (existential versus universal). We illustrate the relevance of a *universal* form of invariants through an example of compositional reasoning.

A current trend in compositional verification of reactive systems is focusing primarily on existential specifications [5,11,16,22–25]. Some notations choose to tailor their semantics so that every trace-based specification becomes existential. In this case, trace-based invariants are equivalent to our $\mathsf{inv}_{\mathcal{E}}$. However, these notations offer no simple way to express universal specifications that are not existential. A key idea of such formalisms is that the *system* `DoorManager` and the *component* `DoorManager` are two different things and should be specified using two different transition systems. The difference between these two systems lies in the way stuttering is implemented. When a closed system is considered, stuttering steps are defined to leave *all* variables unchanged. In our `DoorManager` system from Fig. 1 for instance, there is an implicit stuttering transition equivalent to `doors`, `checkStop`, `speed` := `doors`, `checkStop`, `speed`. If `DoorManager` is a *component* of a larger system instead, the stuttering transition becomes `doors`, `checkStop`, `speed` := `doors`, `checkStop`, $\langle ? \rangle$: the shared variable `speed` is now free to take any value. `DoorManager` can be used as a component only if it is implemented using the second form. If the first form is used, it is a closed system and composition is not possible.

When such an approach is used, a component is associated with a transition system in such a way that the resulting computations are not computations of the component, but instead computations of any system that contains this component. A trace-based specification is no longer a specification of the component. It is already a specification of

a system that uses this component. Not too surprisingly, many difficulties related to composing specifications then disappear naturally. One might even argue that there is no composition involved, as all "component" specifications always refer to the global system that contains these components. In this case, even trace-based invariants are compositional, since they are defined in terms of the computations of a global system. In formalisms that follow this approach, all specifications are compositional in the sense of (EXIST):

$$\mathsf{Spec}.F \;\Rightarrow\; \mathsf{Spec}.(F\|G). \qquad \text{(EXIST)}$$

In particular, any trace-based invariant of F is a trace-based invariant of $F\|G$, since it is defined on the traces of $F\|G$.

Although there are many interesting specifications for which existential composition makes sense, there are also cases where they do not fit very well [26]. Even if trace-based invariants can be composed according to (EXIST), $\mathsf{inv}_{\mathcal{E}}.(\texttt{doors} = \texttt{open} \;\Rightarrow\; \texttt{speed} = 0)$ cannot be used to specify the desired safety property of `DoorManager` for instance. The reason is that it is impossible[4] to implement a Door Manager that satisfies it in a formalism where (EXIST) is valid for all specifications without requiring that this component also has exclusive control over the speed of the train, which would break down compositionality. In a symmetric way, the `Engine` component cannot guarantee this specification either, since it does not have control over the doors.

Notations that rely on (EXIST) for compositional reasoning can deal with this problem by using more elaborate specifications than invariants. For instance, the desired invariant can be replaced with two component specifications, one for the Door Manager and one for the Engine:

$$\mathsf{Spec}_D \triangleq (\texttt{doors} = \texttt{closed}) \wedge \Box(((\texttt{speed} = 0 \vee \texttt{doors} = \texttt{open}) \wedge \texttt{speed}' = \texttt{speed}) \vee (\texttt{doors}' = \texttt{doors}))$$

$$\mathsf{Spec}_E \triangleq (\texttt{speed} = 0) \wedge \Box(((\texttt{speed} \neq 0 \vee \texttt{doors} = \texttt{closed}) \wedge \texttt{doors}' = \texttt{doors}) \vee (\texttt{speed}' = \texttt{speed}))$$

Each specification is written as a formula of temporal logic with binary predicates, where primed variables refer to values in the "next state" [4,27]. The first part of each specification specifies possible initial states, and the second part specifies possible transitions.

Compositional reasoning is achieved by first proving that component implementations, such as $\texttt{Engine}'$ or $\texttt{DoorManager}'$, satisfy Spec_E and Spec_D. Then, because we are in a formalism where all specifications are existential, the composed system satisfies $\mathsf{Spec}_E \wedge \mathsf{Spec}_D$, from which the desired property can be derived. Overall, this verification requires more effort than the corresponding proof using $\mathsf{inv}_{\mathcal{U}}$, both before composition (because the verification of Spec_E and Spec_D involves tricky refinement proofs) and after composition (because one now needs to prove $\mathsf{Spec}_E \wedge \mathsf{Spec}_D \Rightarrow \mathsf{inv}_{\mathcal{W}}.(\texttt{doors} = \texttt{open} \Rightarrow \texttt{speed} = 0)$). Moreover, from the standpoint of a system designer, deriving such component specifications from the desired global property is far from obvious. In particular, it is very difficult to make sure that Spec_E and Spec_D are general enough to not rule out possible correct component implementations.

Instead, we advocate the use of a high-level form of invariants that compose according to rules similar to those of low-level inductive invariants. Although we acknowledge the importance of existential specifications and have indeed based entire case studies on them [17], we also feel that other forms of composition can lead to better specifications in some cases. Universal specifications, in particular, seem to be especially important. The rule (UNIV) is indeed what first comes to mind when composition of invariants is discussed.

In this article, we have introduced $\mathsf{inv}_{\mathcal{U}}$, a form of abstract invariant specifications that are compositional for universal composition. Proof rules for $\mathsf{inv}_{\mathcal{U}}$ and for the usual trace-based (non-compositional) invariants are very similar. There are indeed a number of theorems related to $\mathsf{next}_{\mathcal{U}}$ and $\mathsf{inv}_{\mathcal{U}}$ that are identical to their counterparts for $\mathsf{next}_{\mathcal{W}}$ and $\mathsf{inv}_{\mathcal{W}}$ and that were not mentioned here. The process of verifying an $\mathsf{inv}_{\mathcal{U}}$ specification is not very different from proving an $\mathsf{inv}_{\mathcal{W}}$ specification. Indeed, the idea behind $\mathsf{inv}_{\mathcal{U}}$ is that *some* trace-based invariants can be used in composition, even if they are not inductive. Basically, a trace-based invariant can be composed if the auxiliary invariants used in its proof cannot be challenged by an external environment. The definition of $\mathsf{inv}_{\mathcal{U}}$ is obtained by replacing the traditional notion of strongest invariant (SI) with a form of strongest invariant (SIC) that is not sensitive to a component's environment.

This goes back to an old idea in compositional reasoning, namely the notion of non-interference in proofs, as found in the so-called Owicki–Gries method [28] and applied to compositional reasoning [29,30]. Intuitively, new

[4] More precisely, the only possible implementation would be a Door Manager that *never* opens the doors, which obviously is unacceptable.

components in the environment of a system should not interfere with the way a specification is proved on that system to ensure that this specification is not compromised. Non-interference with the specification itself is not enough in general. Our definition of $\mathsf{inv}_{\mathcal{U}}$ can be seen as a translation, at a semantic level (i.e., independently from proof systems), of this principle.

In [16], a form of strongest invariant similar to our SIC is defined. However, it is only used to define a variety of invariants that correspond to our $\mathsf{inv}_{\mathcal{E}}$ (using a rule similar to $(8_{\mathcal{E}})$). There are no universal invariants. In [24], two forms of invariants are defined that correspond to our $\mathsf{inv}_{\mathcal{E}}$ and $\mathsf{inv}_{\mathcal{U}}$, but not in terms of a strongest invariant. Moreover, the universal form of invariant is defined only as a step in the construction of the existential form, which is the only form used to specify components. Both works are mixing these invariant specifications with complex assumption–commitment rules and, in the case of [24], with refinement rules. It is interesting to note that in both works, the starting point in the definition of compositional specifications is a transition system and its computations. Our method is different, as transitions and computations are completely absent from our definition of $\mathsf{inv}_{\mathcal{U}}$ (although they are used, obviously, in the definition of $\mathsf{inv}_{\mathcal{S}}$ and $\mathsf{inv}_{\mathcal{W}}$). Our approach to making invariant specifications compositional is purely semantic and relies on a predicate transformer WE that is defined in a more general context (i.e., independently from transition systems and temporal logic).

The family of specifications $\mathsf{inv}_{\mathcal{E}}$ is defined by direct application of WE to $\mathsf{inv}_{\mathcal{W}}$. As a result, we know that $\mathsf{inv}_{\mathcal{E}}.p$ is the weakest existential specification stronger than $\mathsf{inv}_{\mathcal{W}}.p$. If we could define a predicate transformer WU such that WU.Spec is the weakest universal specification stronger than Spec, we would define $\mathsf{inv}_{\mathcal{U}}$ as WU.$\mathsf{inv}_{\mathcal{W}}$ and this would guarantee that $\mathsf{inv}_{\mathcal{U}}.p$ is the weakest universal specification stronger than $\mathsf{inv}_{\mathcal{W}}.p$. Unfortunately, some specifications do not have a weakest universal strengthening, and therefore, there is no WU transformer that achieves for universal composition what WE does for existential composition. It may still be the case that $\mathsf{inv}_{\mathcal{U}}.p$ is the weakest universal specification stronger than $\mathsf{inv}_{\mathcal{W}}.p$, but this question is as yet unanswered. Actually, we do not know whether a weakest universal specification stronger than $\mathsf{inv}_{\mathcal{W}}.p$ exists at all.

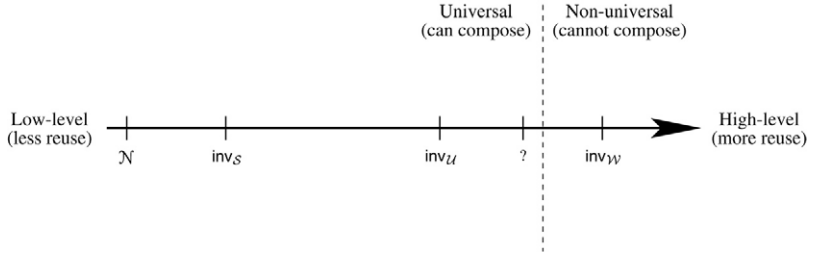

Fig. 6. Weakest universal invariant?

As illustrated in Fig. 6, $\mathsf{inv}_{\mathcal{U}}$ is indeed weaker than $\mathsf{inv}_{\mathcal{S}}$,[5] stronger than $\mathsf{inv}_{\mathcal{W}}$, and encompasses enough information about a component to be universal. What we would like to know is whether there is a better (weaker, more abstract) form of invariant that is also universal.[6]

The approach to building compositional specifications that is presented in this article applies directly to inv and next specifications. A large class of useful safety specifications can be expressed in terms of next, including stability properties and *unless* (or *weak until*) specifications. Liveness and progress specifications, on the other hand, have not been considered at all. The argument has been made that liveness specifications are more difficult to compose than safety specifications. However, although leads-to specifications are notoriously difficult to compose [5,31–34], a property like transient [3,18] composes very well (it is existential) and is the basis for the definition of liveness specifications in UNITY and Seuss. However, transient is not as abstract as leads-to (like $\mathsf{inv}_{\mathcal{S}}$, it refers explicitly to the atomic transitions of a component). What makes leads-to difficult to compose may well be that it is an abstract specification. For instance, the specification of a component responsible for resource allocation might include a property of fair access to resources (expressed as a leads-to). But it could also be specified in terms of a property of absence of deadlock (expressed as an invariant), from which fair access to resources can be

[5] As shown in Section 6, $\mathsf{inv}_{\mathcal{U}}$ is also weaker than $\mathsf{inv}_{\mathcal{E}}$.

[6] If components have no local variables, SIC reduces to *true* and $\mathsf{inv}_{\mathcal{U}}$ is equivalent to $\mathsf{inv}_{\mathcal{S}}$. In this case, it can be shown that it is indeed the weakest universal specification stronger than $\mathsf{inv}_{\mathcal{W}}$.

derived *after* composition. The absence of deadlock specification is less abstract than the fair access specification: For instance, the leads-to specification allows a resource allocator to be implemented through a mechanism that resolves deadlocks a posteriori while the invariant-based specification does not. Moreover, by being more abstract, the leads-to specification requires less verification effort after components are assembled, which is particularly important in a context of reusable components. The main challenge might well be to compose high-level, abstract specifications, whether they are safety or liveness properties.

Our paper focuses on invariants in the context of reactive systems and temporal logic. Another interesting question is that of the case of invariants in an object-oriented programming context (class invariants). The notions of inductive invariants (true before and after each method call, considered individually and atomically) and trace-based invariants (true of the data fields of an object in between method calls at any point during its lifetime) both still exist. They can still be related by a form of strongest invariant that characterizes the reachable states of an object. The argument has been made that the first kind of invariant is preferable because they can be composed more easily [35]. A counter-argument is that they are too low level to be a good specification tool. The laws of composition that are involved there, however, can be more complex than interleaving of atomic transitions [36]. We have started to investigate the generalization of the approach followed in this article for this object-oriented context.

Acknowledgments

The author wishes to thank the anonymous referees for their careful reading of this article and their useful comments. I also extend my thanks to Jayadev Misra, Gérard Padiou, Larry Paulson and Yih-Kuen Tsay for their remarks and the ensuing fruitful discussions I have had with them. I am indebted to Tsay for his help in studying the train example in the context of existential specifications and coming up with various instances of specifications Spec_E and Spec_D.

References

[1] K.M. Chandy, J. Misra, Parallel Program Design: A Foundation, Addison-Wesley, 1988.
[2] J. Misra, A logic for concurrent programming: Safety, Journal of Computer and Software Engineering 3 (2) (1995) 239–272.
[3] J. Misra, A logic for concurrent programming: Progress, Journal of Computer and Software Engineering 3 (2) (1995) 273–300.
[4] L. Lamport, The temporal logic of actions, ACM Transactions on Programming Languages and Systems 16 (3) (1994) 872–923.
[5] M. Abadi, L. Lamport, Conjoining specifications, ACM Transactions on Programming Languages and Systems 17 (3) (1995) 507–534.
[6] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer-Verlag, 1992.
[7] B.A. Sanders, Eliminating the substitution axiom from UNITY logic, Formal Aspects of Computing 3 (2) (1991) 189–205.
[8] E.W. Dijkstra, C.S. Scholten, Predicate Calculus and Program Semantics, in: Texts and Monographs in Computer Science, Springer-Verlag, 1990.
[9] N. Bjørner, I.A. Browne, Z. Manna, Automatic generation of invariants and intermediate assertions, Theoretical Computer Science 173 (1) (1997) 49–87.
[10] A. Pnueli, S. Ruah, L. Zuck, Automatic deductive verification with invisible invariants, in: T. Margaria, W. Yi (Eds.), 7th International Conference on Tools and Algorithms for the Analysis and Construction of Systems, TACAS'2001, in: Lecture Notes in Computer Science, vol. 2031, Springer-Verlag, 2001, pp. 82–97.
[11] B. Finkbeiner, Z. Manna, H.B. Sipma, Deductive verification of modular systems, in: de Roever et al. [37], pp. 239–275.
[12] L. Lamport, Composition: A way to make proofs harder, in: de Roever et al. [37], pp. 402–423.
[13] J. Fiadeiro, T. Maibaum, Verifying for reuse: foundations of object-oriented system verification, in: I.M.C. Hankin, R. Nagarajan (Eds.), Theory and Formal Methods, World Scientific Publishing Company, 1995, pp. 235–257.
[14] M. Charpentier, K.M. Chandy, Specification transformers: A predicate transformer approach to composition, Acta Informatica 40 (3) (2004) 265–301.
[15] K.M. Chandy, B. Sanders, Reasoning about program composition, Technical Report, University of Florida. Available as http://www.cise.ufl.edu/~sanders/pubs/composition.ps, 2000.
[16] P. Collette, Design of compositional proof systems based on assumption–commitment specifications. Application to UNITY, Doctoral Thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, June 1994.
[17] K.M. Chandy, M. Charpentier, An experiment in program composition and proof, Formal Methods in System Design 20 (1) (2002) 7–21.
[18] J. Misra, A discipline of multiprogramming: programming theory for distributed applications, in: Monographs in Computer Science, Springer-Verlag, 2001.
[19] W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, J. Zwiers, Concurrency Verification: Introduction to Compositional and Noncompositional Methods, Cambridge University Press, 2001.
[20] M. Charpentier, An approach to composition motivated by wp, in: R. Kutsche, H. Weber (Eds.), Fundamental Approaches to Software Engineering, FASE'2002, in: Lecture Notes in Computer Science, vol. 2306, Springer-Verlag, 2002, pp. 1–14.
[21] C. Hoare, Communicating Sequential Processes, Prentice-Hall International, 1984.

[22] C. Hoare, A theory of conjunction and concurrency, in: International Conference on Databases, Parallel Architectures and Their Applications, ParBase'90, IEEE, 1990, pp. 304–309.
[23] M. Abadi, L. Lamport, Composing specifications, ACM Transactions on Programming Languages and Systems 15 (1) (1993) 73–132.
[24] R.T. Udink, Program refinement in UNITY-like environments, Ph.D. Thesis, Utrecht University, September 1995.
[25] C. Hoare, J. He, Unifying Theories of Programming, 1st ed., Prentice-Hall, 1998.
[26] M. Charpentier, K.M. Chandy, Examples of program composition illustrating the use of universal properties, in: J. Rolim (Ed.), International Workshop on Formal Methods for Parallel Programming: Theory and Applications, FMPPTA'99, in: Lecture Notes in Computer Science, vol. 1586, Springer-Verlag, 1999, pp. 1215–1227.
[27] B. Jonsson, Y.-K. Tsay, Assumption/guarantee specifications in linear-time temporal logic, Theoretical Computer Science 167 (1–2) (1996) 47–72.
[28] S. Owicki, D. Gries, An axiomatic proof technique for parallel programs, Acta Informatica 6 (1976) 319–340.
[29] J. Misra, The importance of ensuring, in: Notes on Unity, Department of Computer Science, University of Texas at Austin, 1990, note 11.
[30] Q. Xu, W.-P. de Roever, J. He, The rely–guarantee method for verifying shared variable concurrent programs, Formal Aspects of Computing 9 (2) (1997) 149–174.
[31] E. Cohen, Modular progress proofs of concurrent programs, Ph.D. Thesis, University of Texas at Austin, August 1992.
[32] K.M. Chandy, B.A. Sanders, Predicate transformers for reasoning about concurrent computation, Science of Computer Programming 24 (1995) 129–148.
[33] D. Meier, B. Sanders, Composing leads-to properties, Theoretical Computer Science 243 (1–2) (2000) 339–361.
[34] E. Cohen, Asynchronous progress, in: A. McIver, C. Morgan (Eds.), Programming Methodology, Springer-Verlag, 2003, pp. 57–68.
[35] K.R.M. Leino, G. Nelson, J.B. Saxe, ESC/Java User's Manual, Compaq Systems Research Center, October 2000.
[36] P. Müller, Modular specification and verification of object oriented programs, Ph.D. Thesis, FernUniversität Hagen, in: LNCS, vol. 2262, Springer-Verlag, 2001.
[37] W.-P. de Roever, H. Langmaack, A. Pnueli (Eds.), Compositionality: The Significant Difference. International Symposium, COMPOS'97, in: Lecture Notes in Computer Science, vol. 1536, Springer-Verlag, 1997.